GPU FOR DEEP LEARNING

chandlerz@nvidia.com 周国峰

Wuhan University 2017/10/13

NVIDIA

Agenda

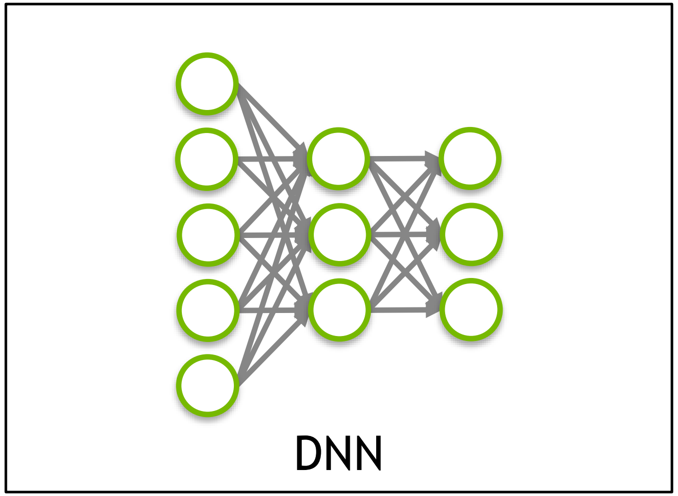
Why Deep Learning Boost Today?

Nvidia SDK for Deep Learning?

- CUDA 8.0
- cuDNN
- TensorRT (GIE)
- NCCL
- DIGITS

Why Deep Learning Boost Today?

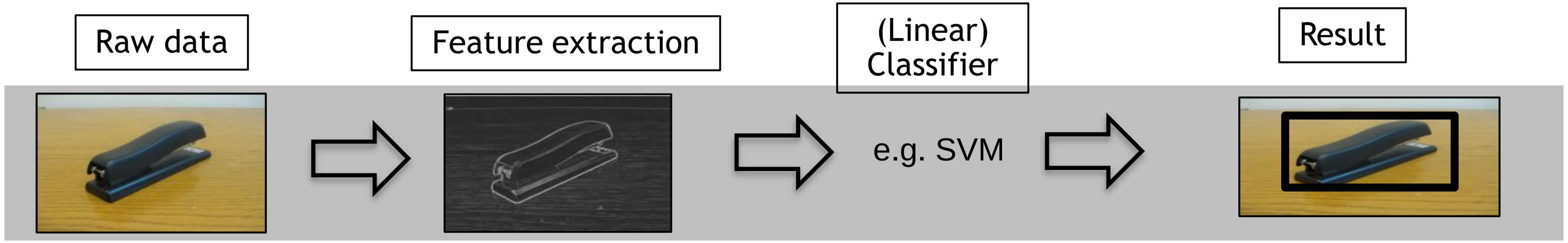
WHY DEEP LEARNING BOOST TODAY?



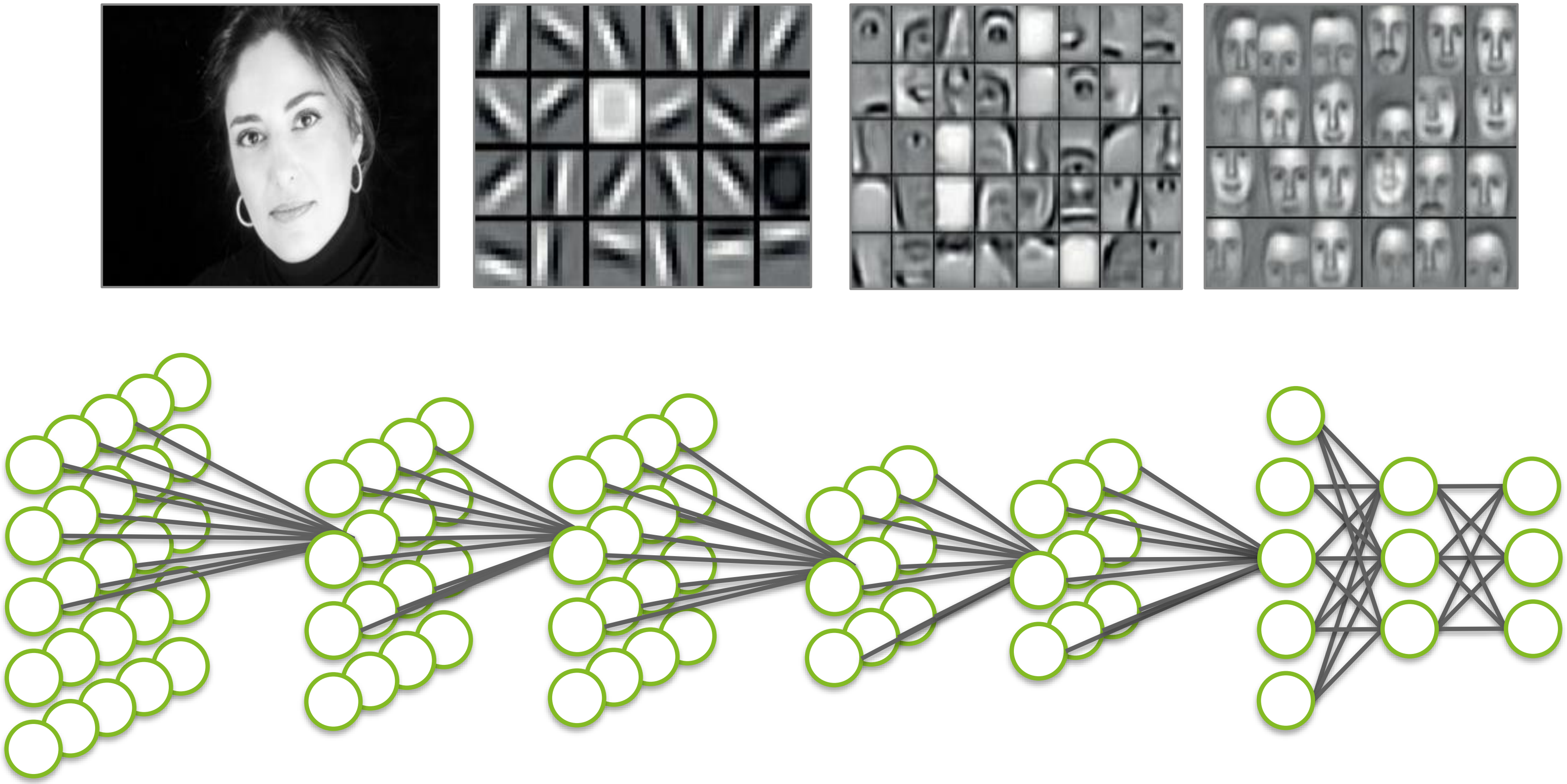
"Google's AI engine also reflects how the world of computer hardware is changing. (It) depends on machines equipped with GPUs… And it depends on these chips more than the larger tech universe realizes."

WIRED

TRADITIONAL COMPUTER VISION APPROACH

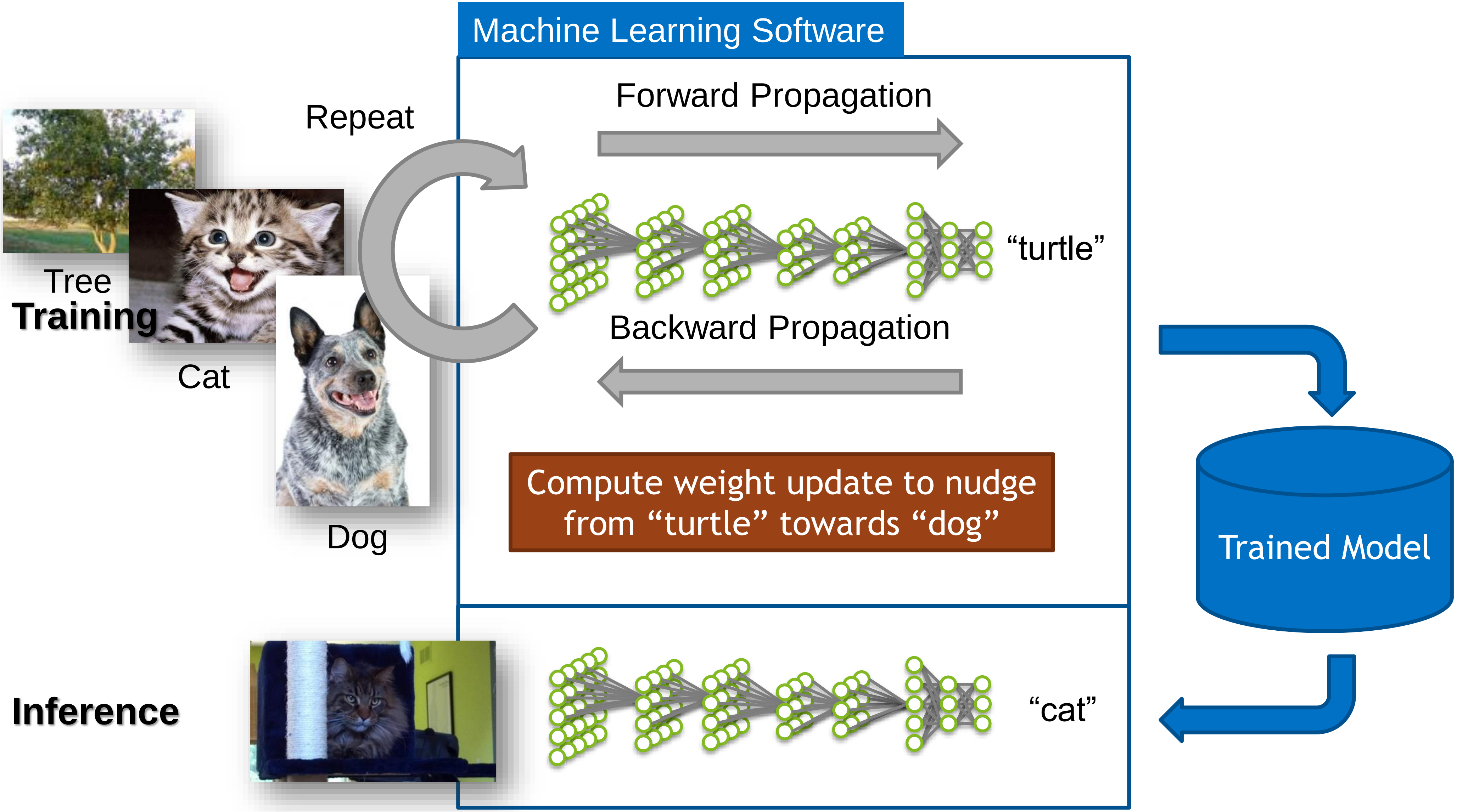


WHAT IS DEEP LEARNING?



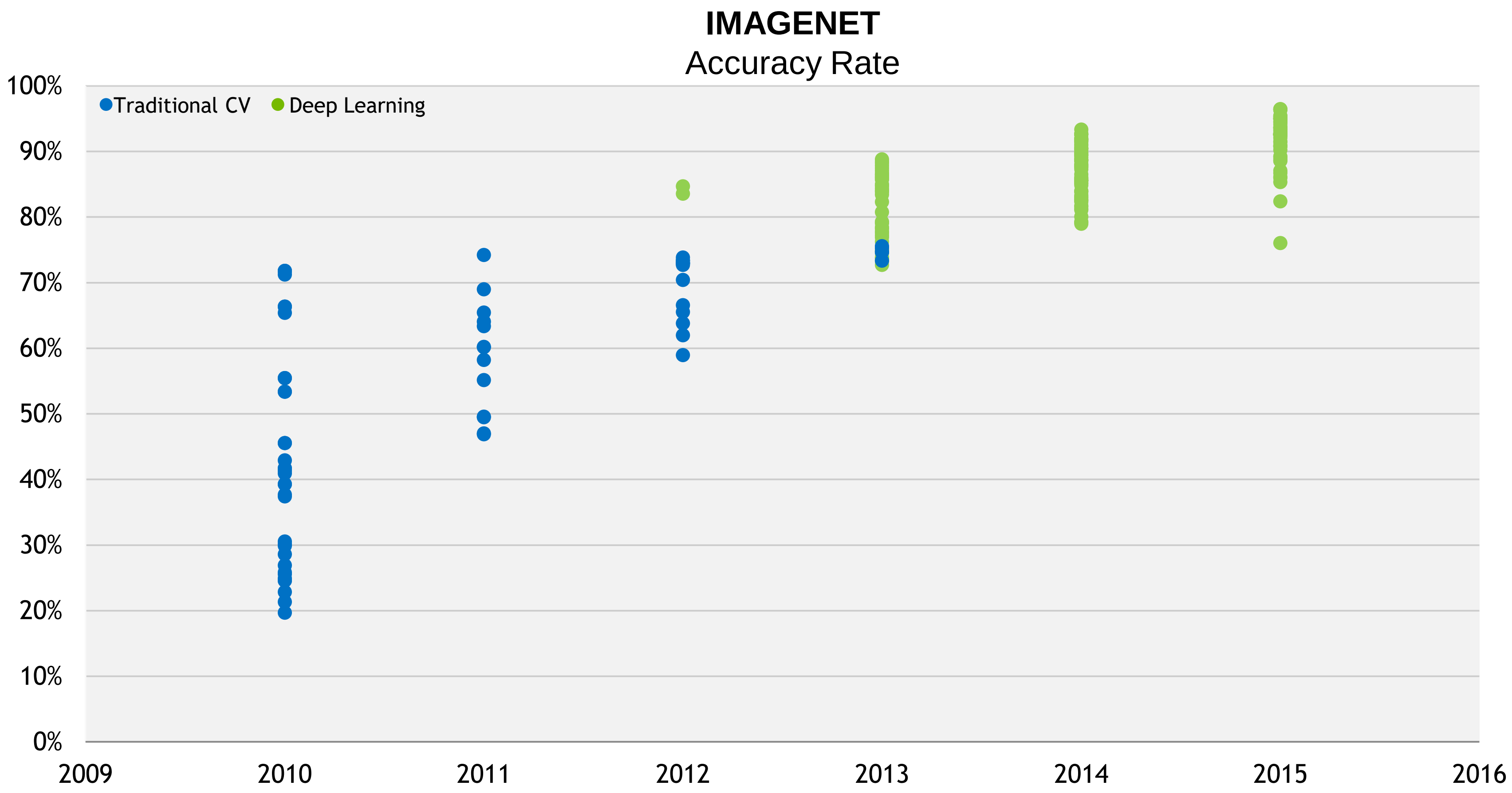
Typical Network

Task objective
e.g. Identify face
Training data
10-100M images
Network architecture
10 layers
1B parameters
Learning algorithm
~30 Exaflops
~30 GPU days


Machine Learning Software
Forward Propagation
Repeat
"turtle"
Tree
Training
Cat
Backward Propagation
Dog
Compute weight update to nudge from "turtle" towards "dog"
Trained Model
Inference
"cat"

DEEP LEARNING FOR VISUAL PERCEPTION

Going from strength to strength



Deep learning with COTS HPC systems

A. Coates, B. Huval, T. Wang, D. Wu, A. Ng, B. Catanzaro

ICML 2013

"Now You Can Build Google's $1M Artificial Brain on the Cheap"

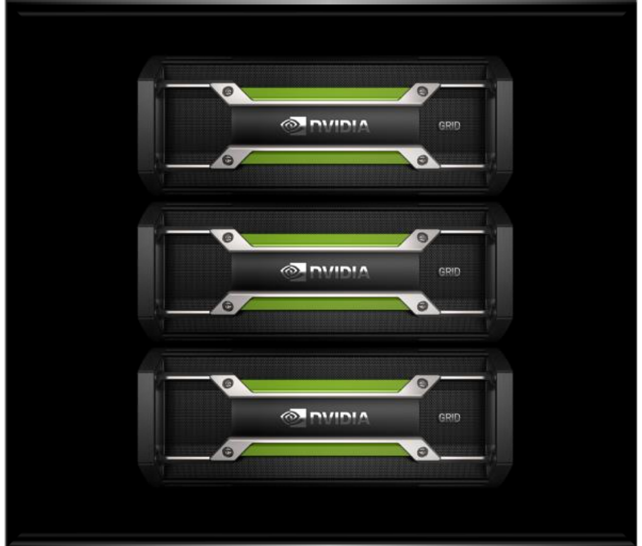
WIRED

GOOGLE DATACENTER

1,000 CPU Servers
2,000 CPUs • 16,000 cores

600 kWatts
$5,000,000

STANFORD AI LAB



3 GPU-Accelerated Servers
12 GPUs • 18,432 cores

4 kWatts
$33,000

EXABYTES OF CONTENT PRODUCED DAILY

User-Generated Content Dominates Web Services

10M Users 40 years of video/day	1.7M Broadcasters Users watch 1.5 hours/day	6B Queries/day 10% use speech
Periscope	twitch	Baidu 百度
270M Items sold/day 43% on mobile devices	8B Video views/day 400% growth in 6 months	300 hours of video/minute 50% on mobile devices
淘宝网 Taobao.com	facebook	YouTube

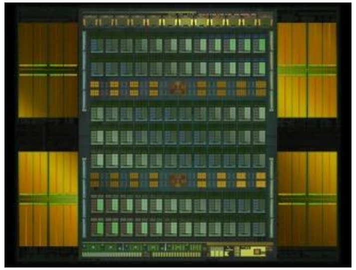
NVIDIA SDK for Deep Learning

1. What’s New in CUDA 9.0

INTRODUCING CUDA 9

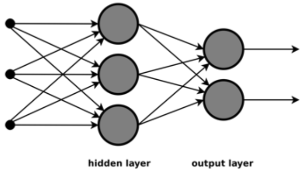
BUILT FOR VOLTA

Tesla V100
New GPU Architecture
Tensor Cores
NVLink
Independent Thread Scheduling



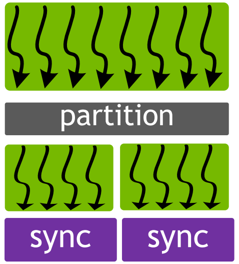
FASTER LIBRARIES

cuBLAS for Deep Learning
NPP for Image Processing
cuFFT for Signal Processing



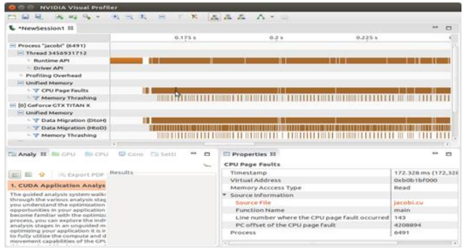
COOPERATIVE THREAD GROUPS

Flexible Thread Groups
Efficient Parallel Algorithms
Synchronize Across Thread Blocks in a Single GPU or Multi-GPUs



DEVELOPER TOOLS & PLATFORM UPDATES

Faster Compile Times
Unified Memory Profiling
NVLink Visualization
New OS and Compiler Support



FASTEST LIBRARIES

VOLTA PLATFORM SUPPORT

Utilize Volta Tensor Cores

Volta optimized GEMMs (cuBLAS)

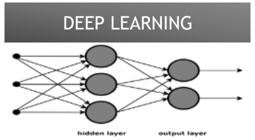
Out-of-box performance on Volta (all libraries)

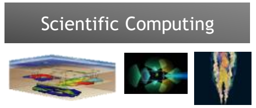
PERFORMANCE

GEMM optimizations for RNNs (cuBLAS)

Faster image processing (NPP)

FFT optimizations across various sizes (cuFFT)

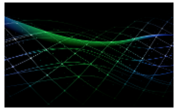


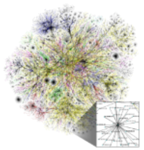


NEW ALGORITHMS

Multi-GPU dense & sparse solvers, dense eigenvalue & SVD (cuSOLVER)

Breadth first search, clustering, triangle counting, extraction & contraction (nvGRAPH)





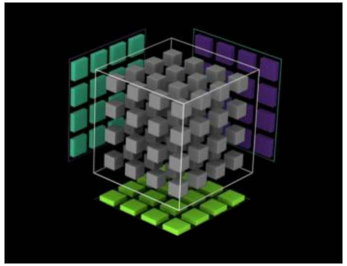
IMPROVED USER EXPERIENCE

New install package for CUDA Libraries (library-only meta package)

Modular NPP with small footprint, support for image batching

TENSOR CORE

120 Tensor TFLOPS DL core



GV100:
80 Volta SM
 * 8 Tensor Core/SM
 * 128 Tensor FLOPs/Tensor Core/clock
 * 1462 MHz
≃ 120 Tensor TFLOPS

V.S.

In the movie Terminator III:
Skynet is said to be
operating at "60 teraflops
per second."

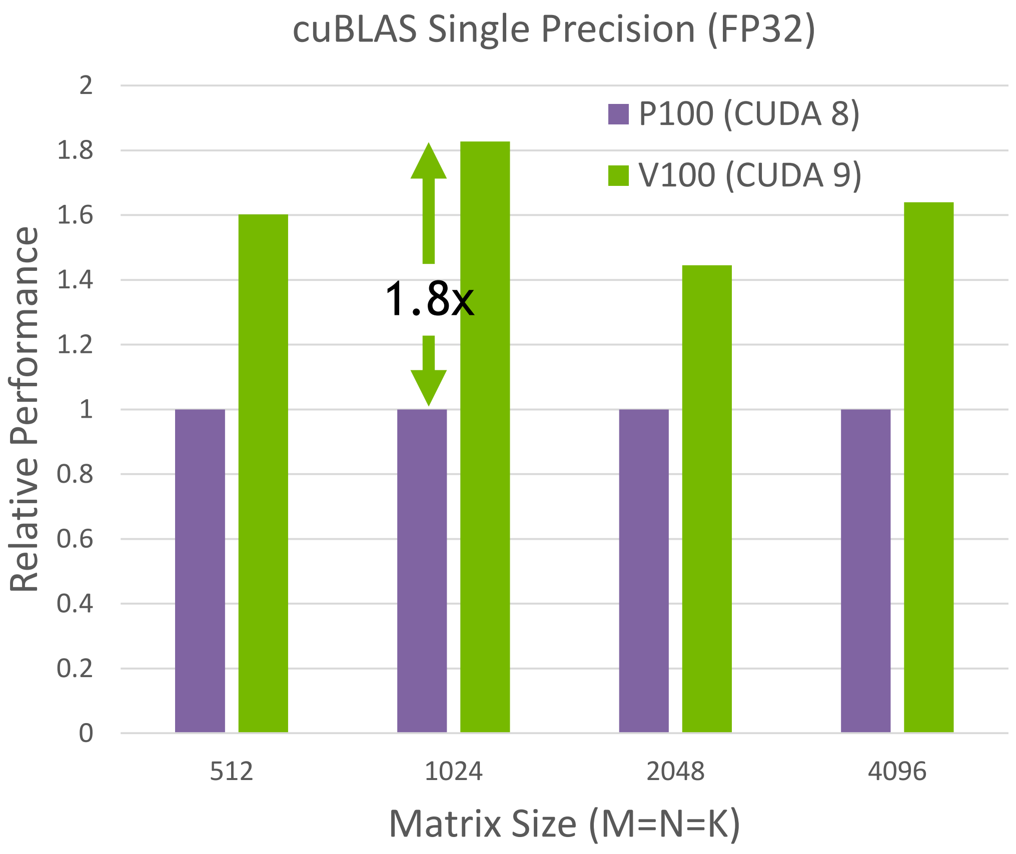
$$
D = \begin{pmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix} \begin{pmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{pmatrix} + \begin{pmatrix} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{pmatrix}
$$

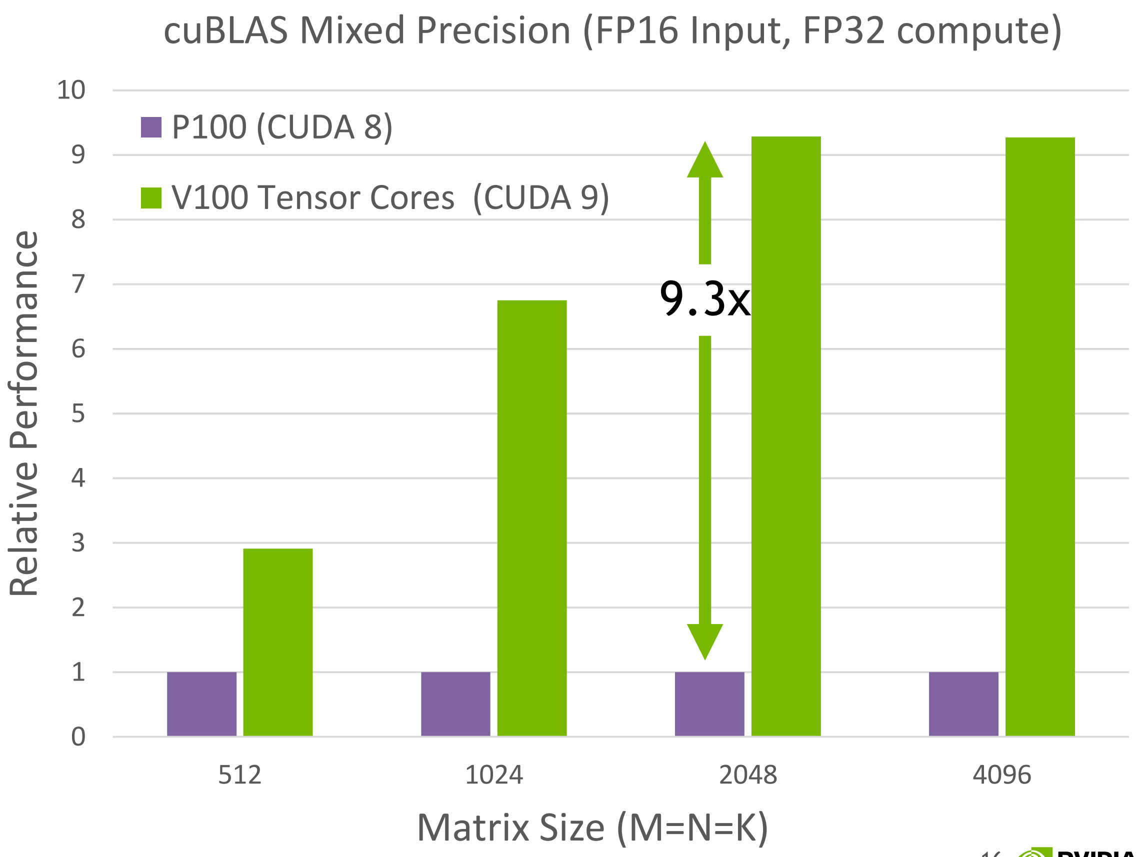
FP16 or FP32 | FP16 | FP16 | FP16 or FP32

$$D = AB + C$$

cuBLAS GEMMS FOR DEEP LEARNING

V100 Tensor Cores + CUDA 9: over 9x Faster Matrix-Matrix Multiply

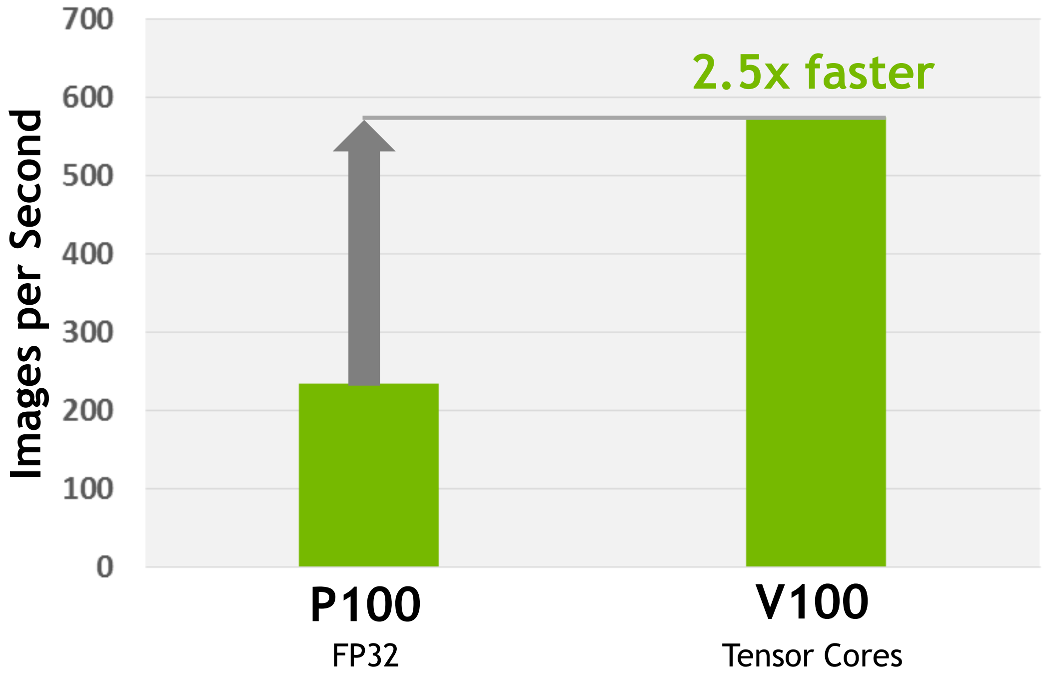




Note: pre-production Tesla V100 and pre-release CUDA 9. CUDA 8 GA release.

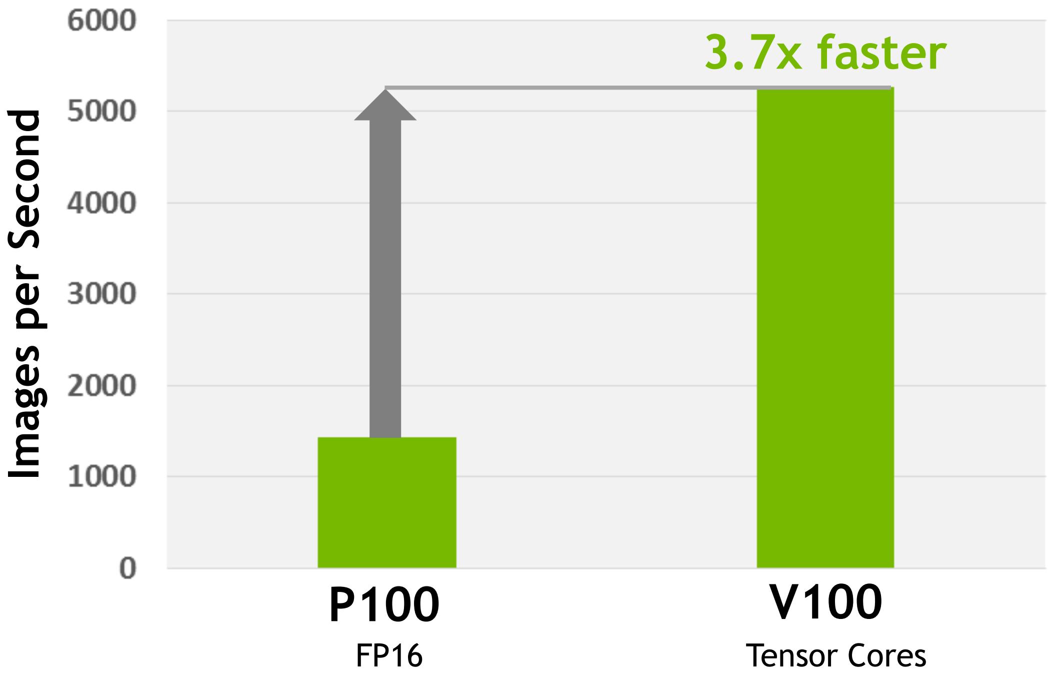
VOLTA: A GIANT LEAP FOR DEEP LEARNING

ResNet-50 Training



ResNet-50 Inference

TensorRT - 7ms Latency



V100 measured on pre-production hardware.

cuDNN

cuDNN: Design Goals

Basic Deep Learning Subroutines

Allow user to write a DNN application without any custom CUDA code

Flexible Layout

Handle any data layout

Memory - Performance tradeoff

Good perf with minimal memory use, great perf with more memory use

CUDNN 7

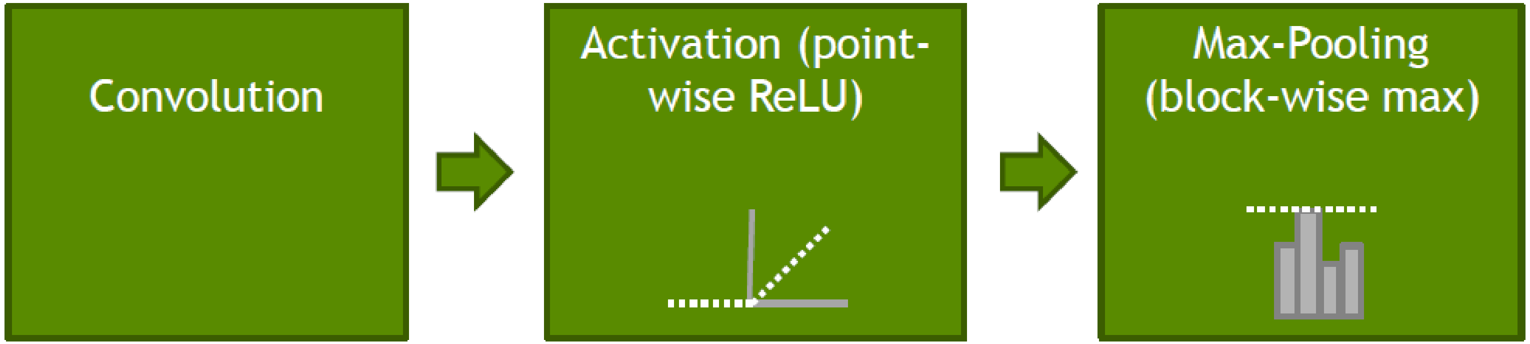
Key Features

- Forward and backward convolution routines, including cross-correlation, designed for convolution neural nets.
- LSTM and GRU Recurrent Neural Networks (RNN) and Persistent RNN.
- Forward and backward paths for many common layer types such as pooling, LRN, LCN, batch normalization, dropout, CTC, ReLU, sigmoid, softmax and tanh.
- Tensor transformation functions.
- Accelerated convolution using FP16, INT8 and mixed-precision Tensor Cores operations on Volta GPUs.

Typical layers in CNN

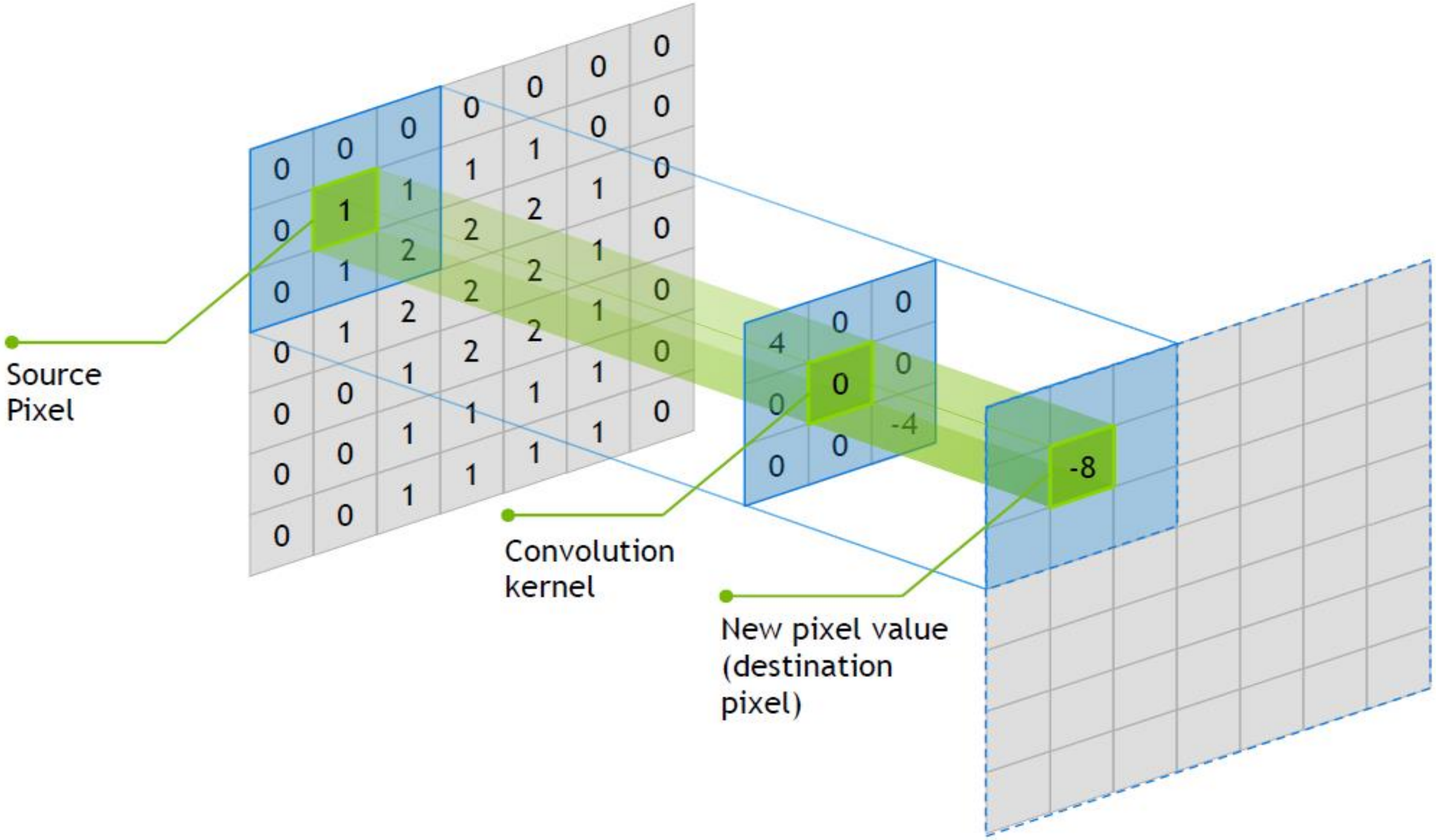
Block of layers is typically made up of 2-3 stages:

- Linear Transformation of data - convolution - 80-90% of execution time
- Activation - Point-wise application of non-linear function
- Optional pooling - Spatial Smoothing



developer.nvidia.com/cudnn

Convolution



cuDNN Example: Convolution

Tensor Definition

cudnnCreateTensorDescriptor to create tensor descriptor

cudnnSetTensorNDDescriptor to set tensor descriptor size

Choose Convolution Algo

cudnnFindConvolutionForwardAlgorithm to choose a convolutional algorithm

Carry out (Forward) Algorithm

cudnnConvolutionForward to do forward convolution

Example with cuDNN

Tensor Definition

```
cudnnDataType_t dataType = CUDNN_DATA_FLOAT;                     // define data type

cudnnTensorFormat_t tensorFormat = CUDNN_TENSOR_NCHW;            // define layout

checkCUDNN( cudnnCreateTensorDescriptor(&tensorDesc) );          // create tensor

checkCUDNN( cudnnSetTensor4dDescriptor(tensorDesc, tensorFormat, // set the tensor
dataType, n, c, h, w) );
```

Example with cuDNN

Set Convolution Parameters

```
checkCUDNN( cudnnCreateFilterDescriptor(&filterDesc) );
checkCUDNN( cudnnCreateConvolutionDescriptor(&convDesc) );
checkCUDNN( cudnnCreateTensorDescriptor(&biasDesc) );

checkCUDNN( cudnnSetFilter4dDescriptor(filterDesc, dataType, k, c, r, s) );
checkCUDNN( cudnnSetConvolution2dDescriptor(convDesc, pad_h, pad_w, hs,
ws, 1, 1, CUDNN_CROSS_CORRELATION) );
checkCUDNN( cudnnSetTensor4dDescriptor(biasDesc, tensorFormat, dataType,
1, k, 1, 1) );
```

Example with cuDNN

Choose Convolution Algo

```
    cudnnFindConvolutionForwardAlgorithm(cudnnHandle, IN.tensorDesc, L.filterDesc,
L.convDesc, OUT.tensorDesc, 10, &algoCount, perfs );

    for(int i=0; i<algoCount; i++) {

        printf("algo: %d, %d, %f ms, %ld B, err: %s\n", i, (int)(perfs[i].algo),
perfs[i].time, perfs[i].memory, cudnnGetErrorString(perfs[i].status));

    }
```

Example with cuDNN

Forward Convolution

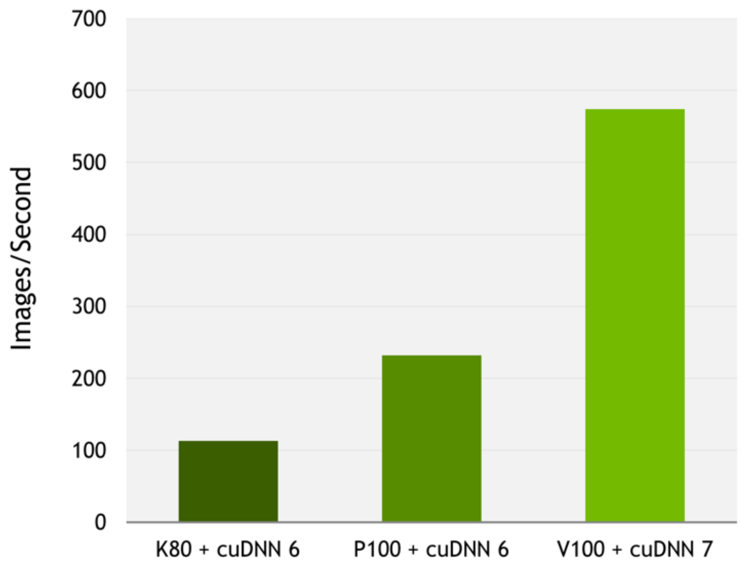
```
float a= 1.0f; float b = 0.0f;

checkCUDNN( cudnnConvolutionForward(cudnnHandle,     &(a),
                                    IN.tensorDesc,    IN.ptr,
                                    L.filterDesc,   L.ptr,
                                    L.convDesc,  L.algo,
                                    L.tmpMemoryPtr, L.memoryInBytes,
                                    &(b),
                                    OUT.tensorDesc,   OUT.ptr) );
```

CUDNN V7 + VOLTA

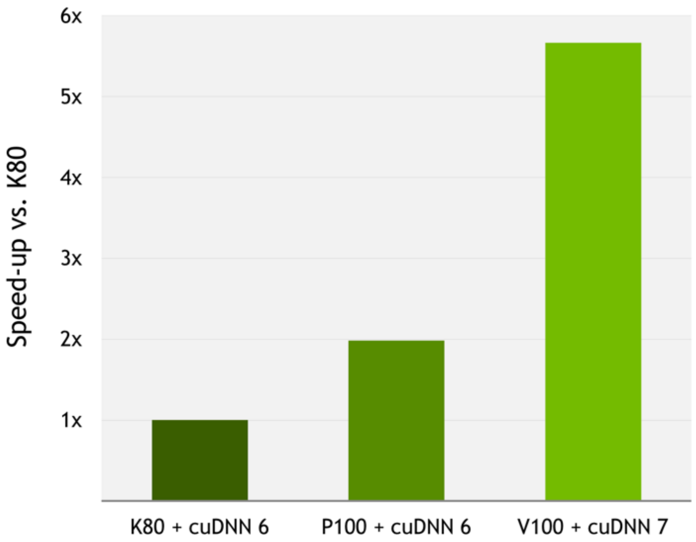
A Giant leap for Deep learning

2.5x Faster Training of CNNs



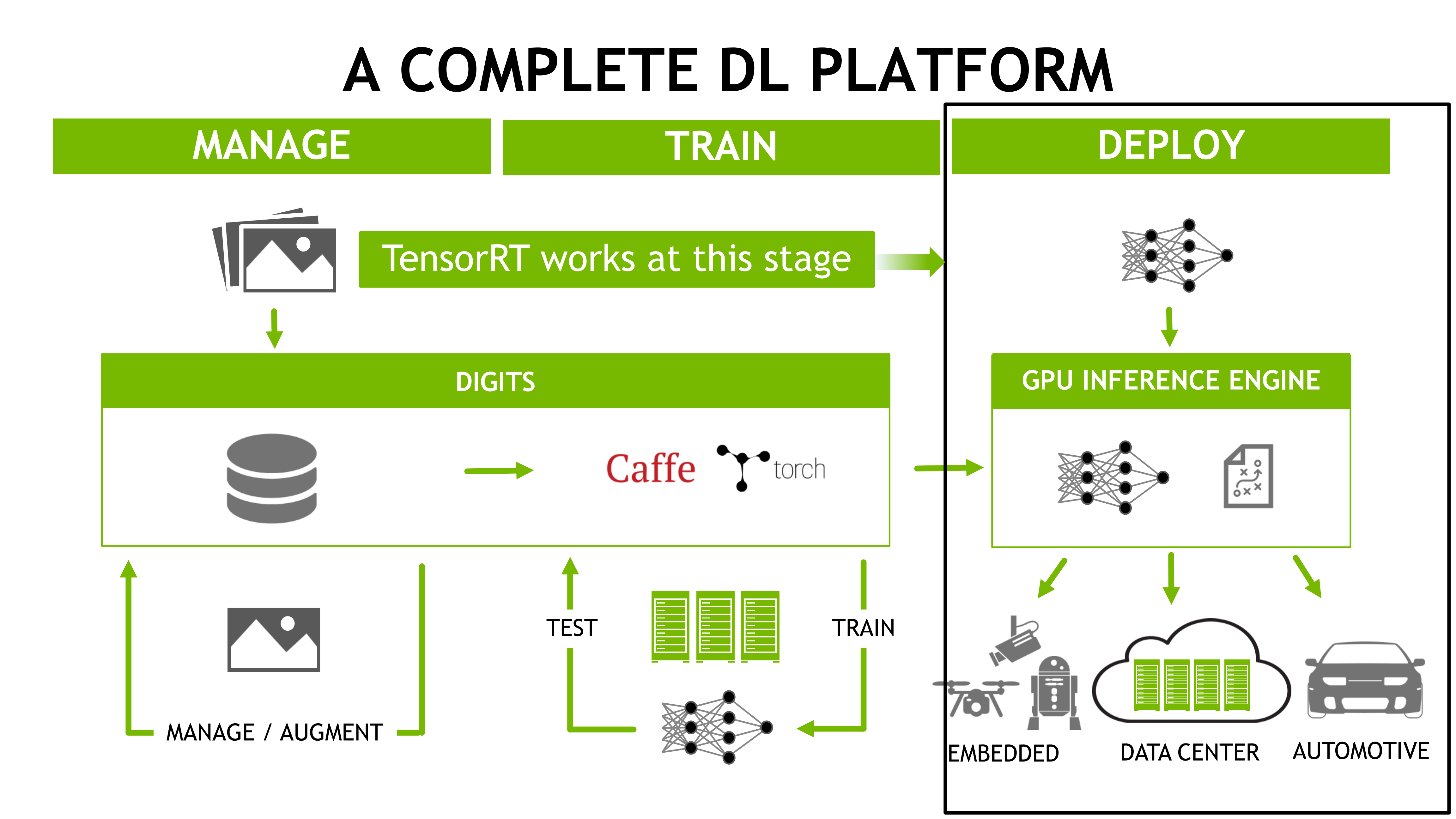
Caffe2 performance (images/sec), Tesla K80 + cuDNN 6 (FP32), Tesla P100 + cuDNN 6 (FP32), Tesla V100 + cuDNN 7 (FP16, pre-release H/W and S/W). ResNet50, Batch size: 64

3x Faster Training of LSTM RNNs



MXNet performance (min/epoch), Tesla K80 + cuDNN 6 (FP32), Tesla P100 + cuDNN 6 (FP32), Tesla V100 + cuDNN 7 (FP16, pre-release H/W and S/W). NMT seq2seq RNN (https://github.com/mkolod/mxnet_seq2seq)

TensorRT (GIE)


A COMPLETE DL PLATFORM
MANAGE
TRAIN
DEPLOY
TensorRT works at this stage
DIGITS
Caffe
torch
GPU INFERENCE ENGINE
TEST
TRAIN
MANAGE / AUGMENT
EMBEDDED
DATA CENTER
AUTOMOTIVE

INFERENCE VS TRAINING

Inference compared with training

Static weights and no back-propagation

- Enable graph optimizations
- Simplify the memory management

Note: No fine-tuning in TensorRT

Smaller batch size

- Harder to achieve high GPU utilization

High speed reduced precision (FP16/INT8)

- Provide opportunities for bandwidth savings and faster calculations

TENSORRT WORKFLOW

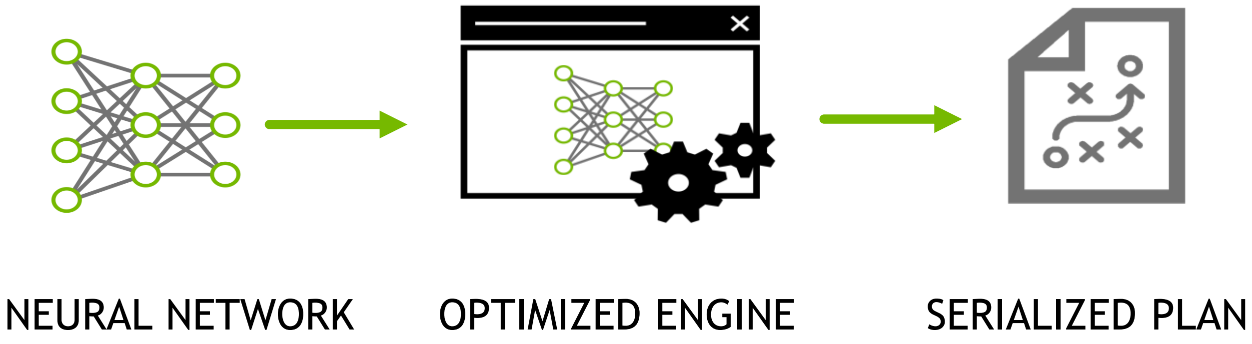
Optimization and serialization

Input

- A pre-trained FP32 model

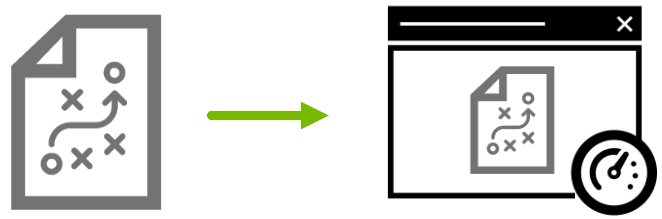
Output

- An optimized execution engine (PLAN) on GPU for serialization



TENSORRT WORKFLOW

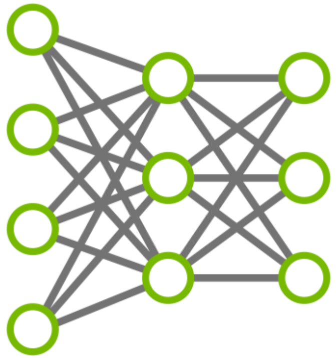
Deployment



Serialized a PLAN can be reloaded from the disk into the TensorRT runtime. There is no need to perform the optimization step again.

TENSORRT

Optimization strategies



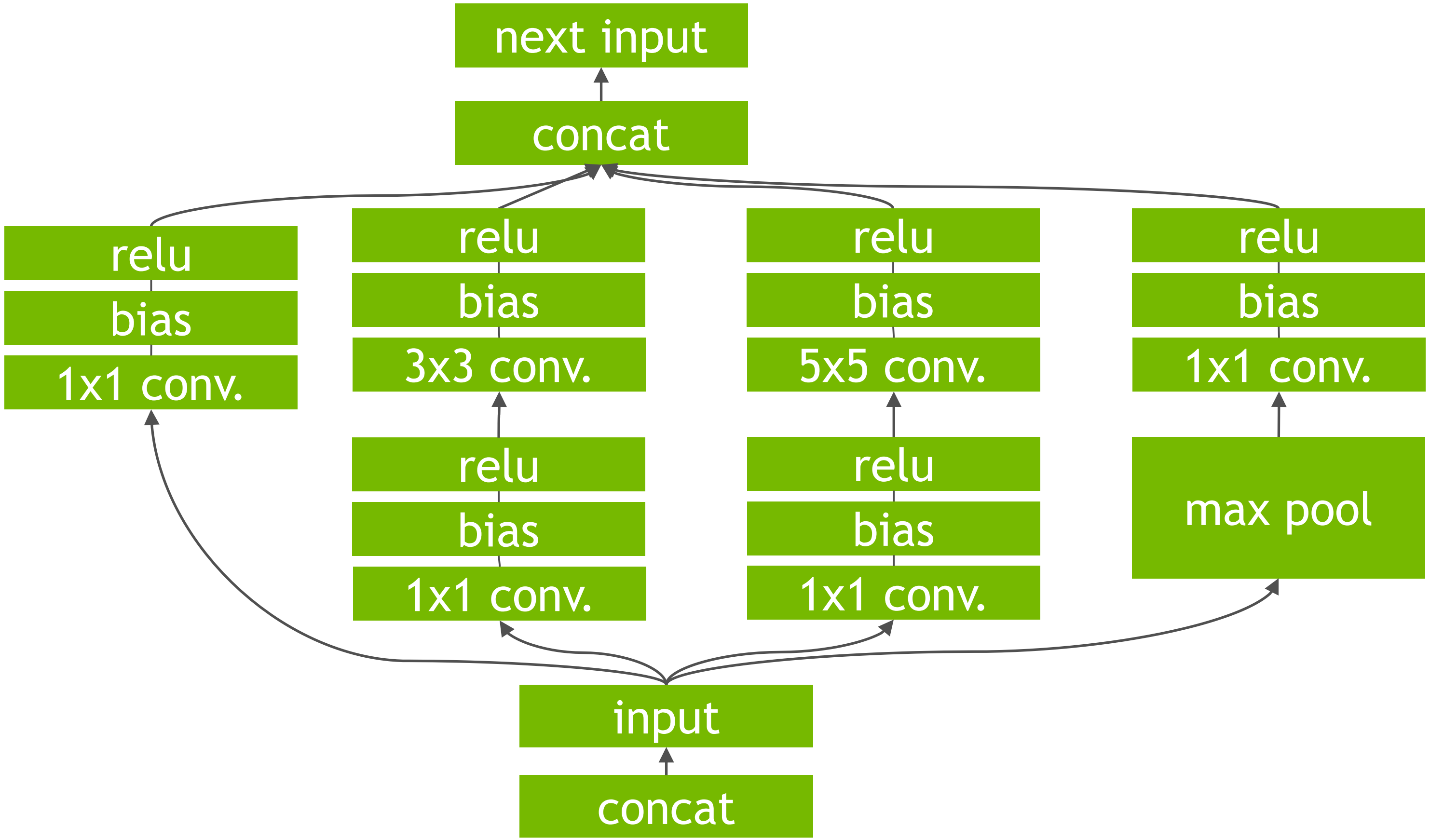
TRAINED
NEURAL NETWORK

- Fuse network layers
- Eliminate concatenation layers
- Kernel specialization
- Auto-tuning for target platforms
- Select optimal tensor layouts
- Batch size tuning
- FP16/INT8 acceleration

OPTIMIZED
INFERENCE
RUNTIME

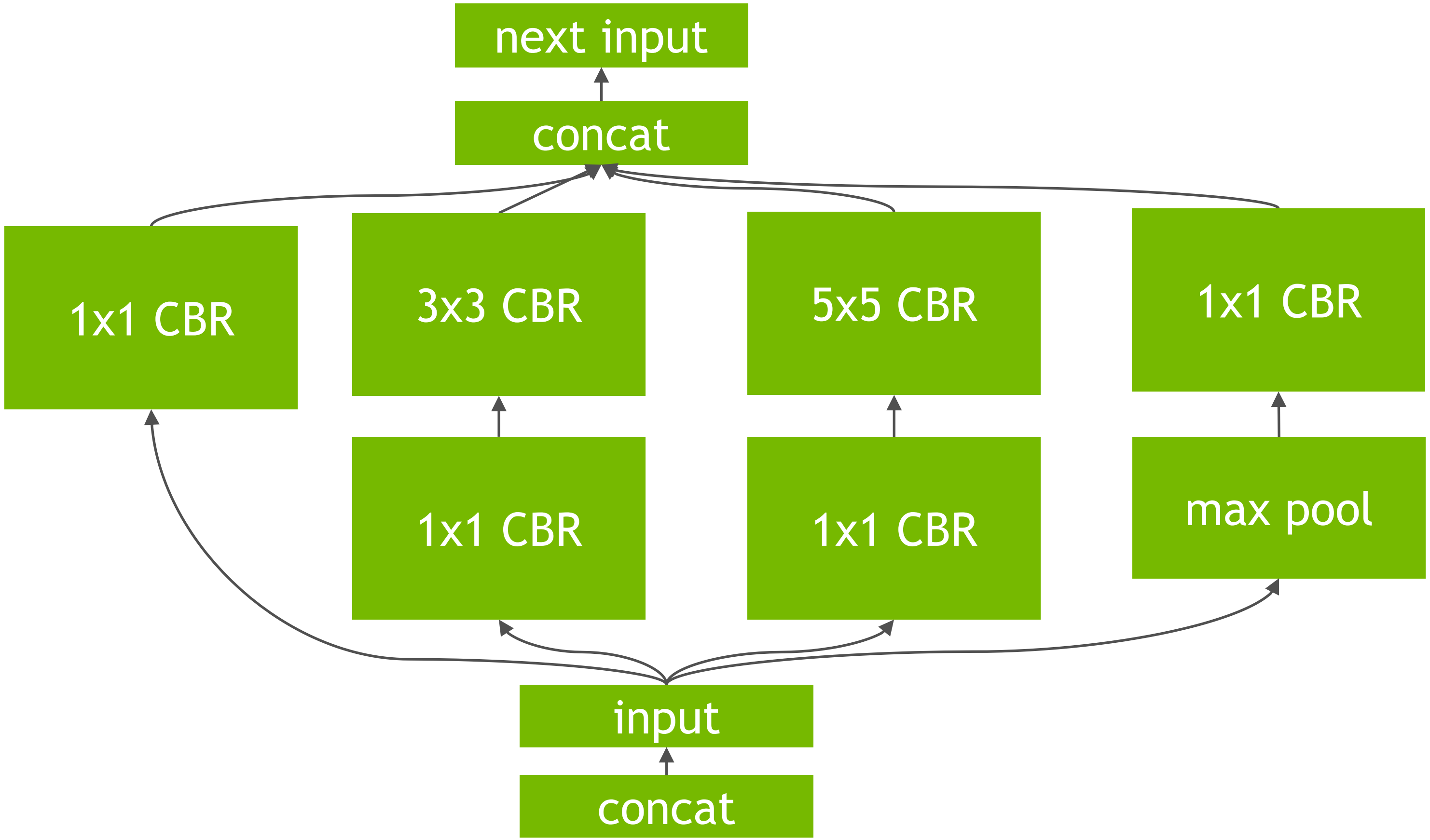
FUSE NETWORK LAYERS

Inception structure in GoogLenet



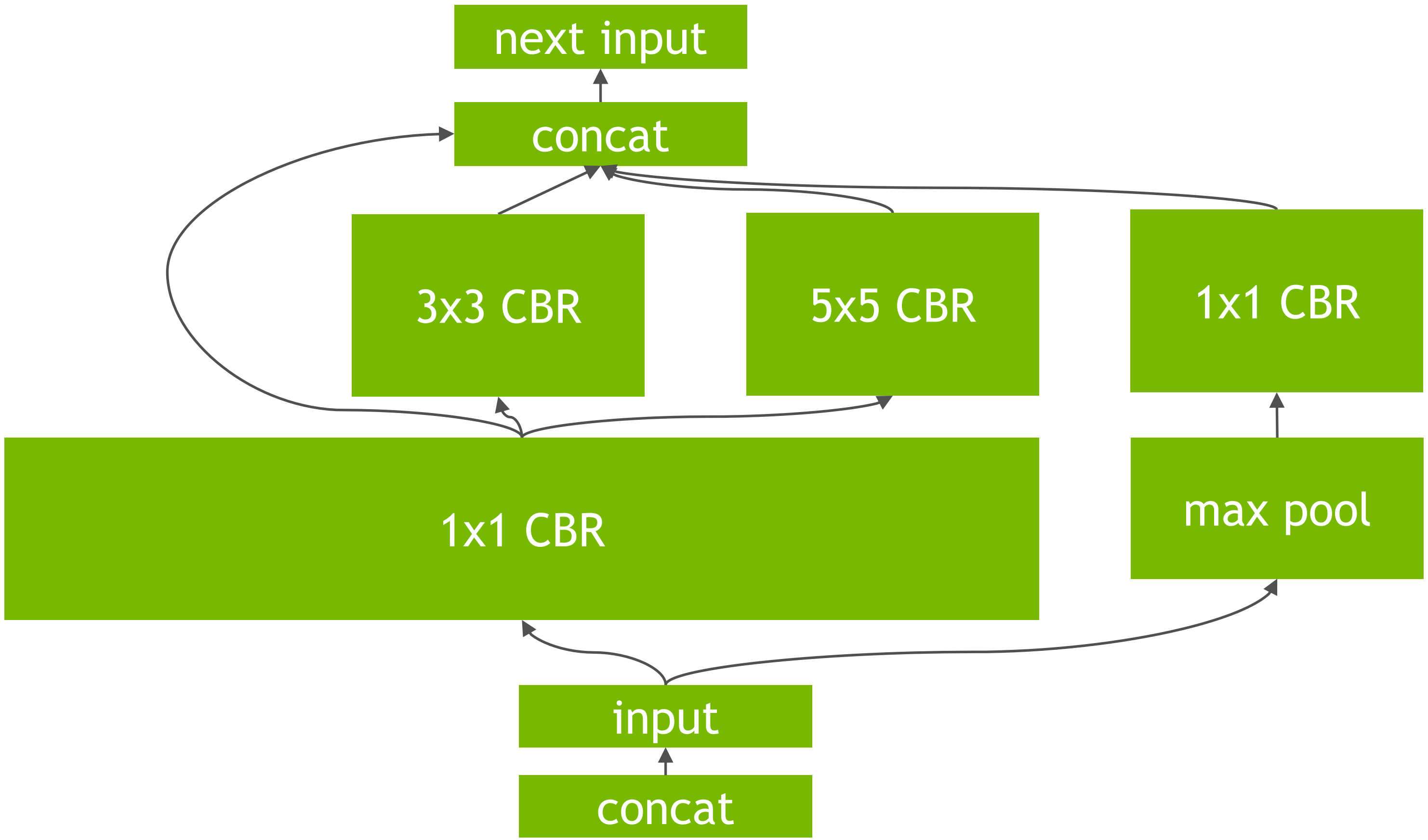
FUSE NETWORK LAYERS

Vertical fusion



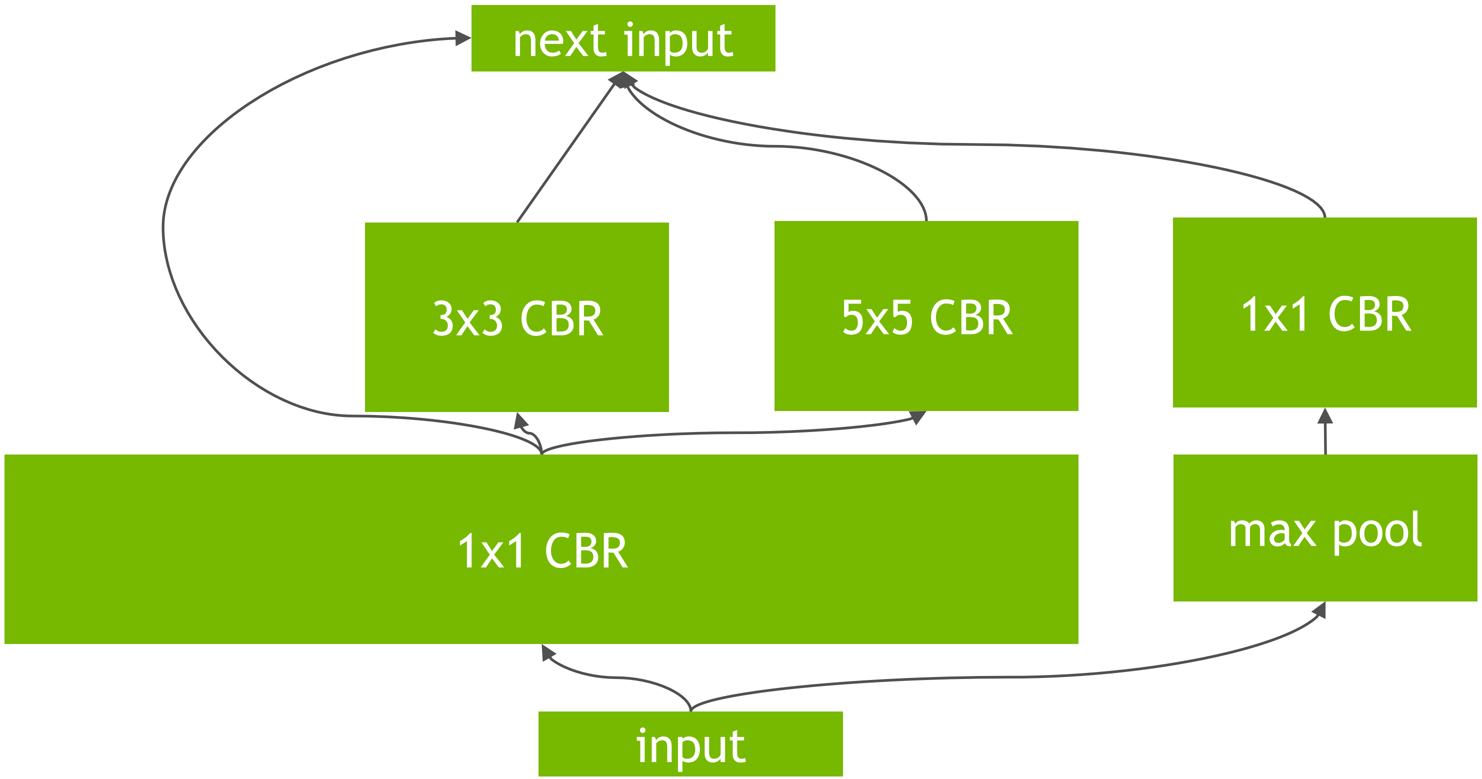
FUSE NETWORK LAYERS

Horizontal fusion



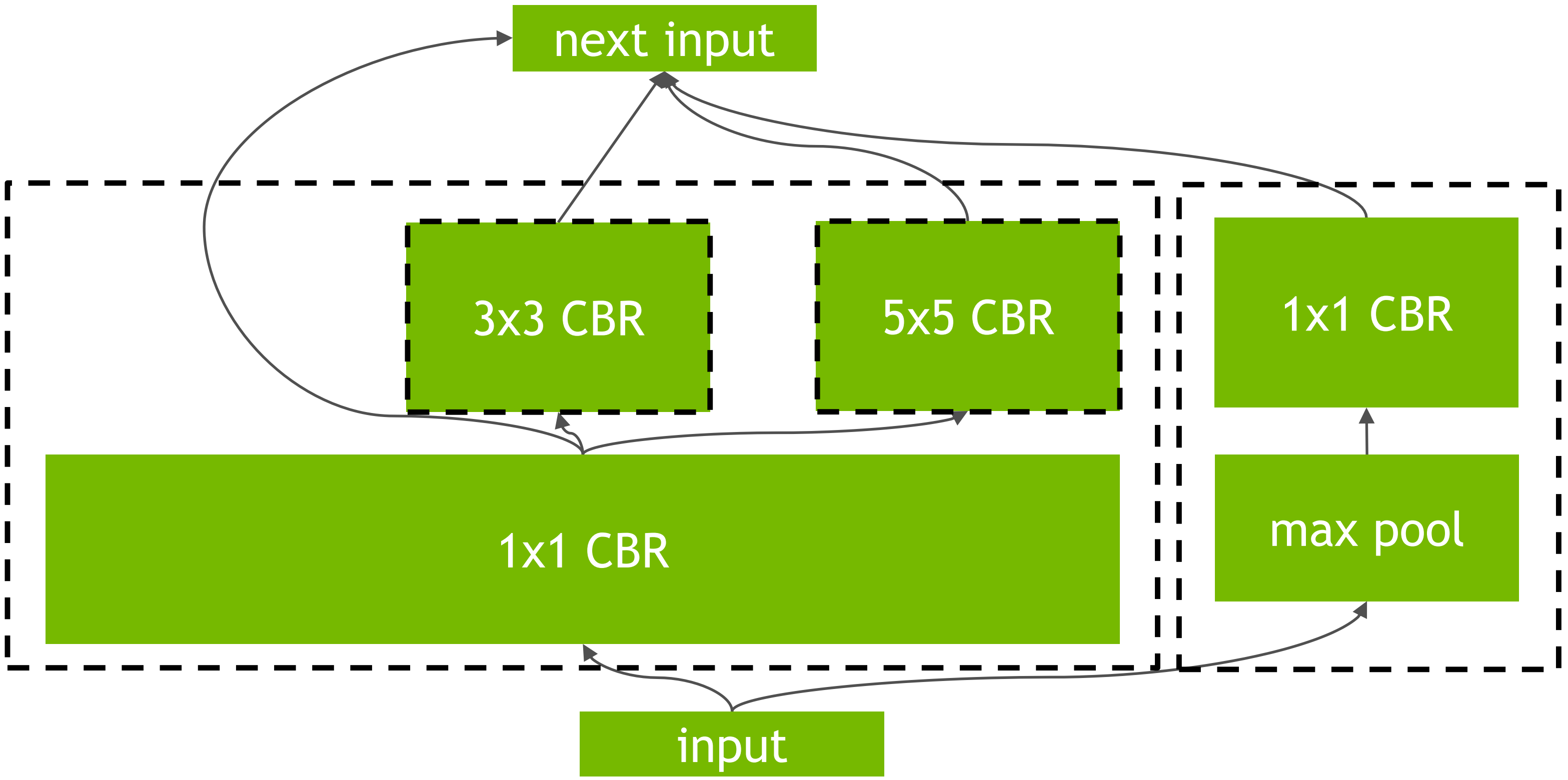
FUSE NETWORK LAYERS

Concat elision



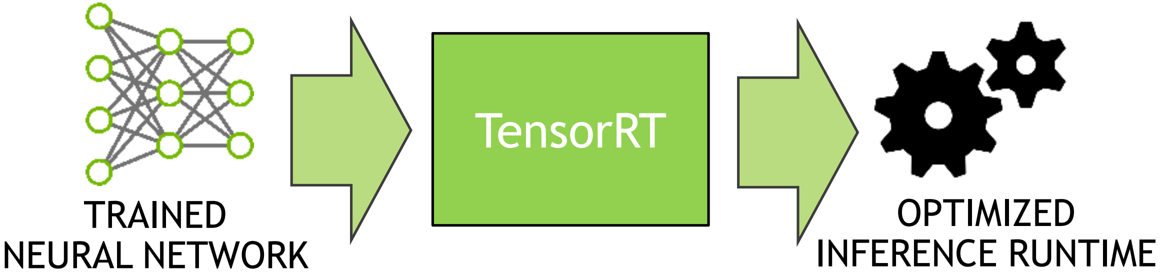
FUSE NETWORK LAYERS

Concurrency

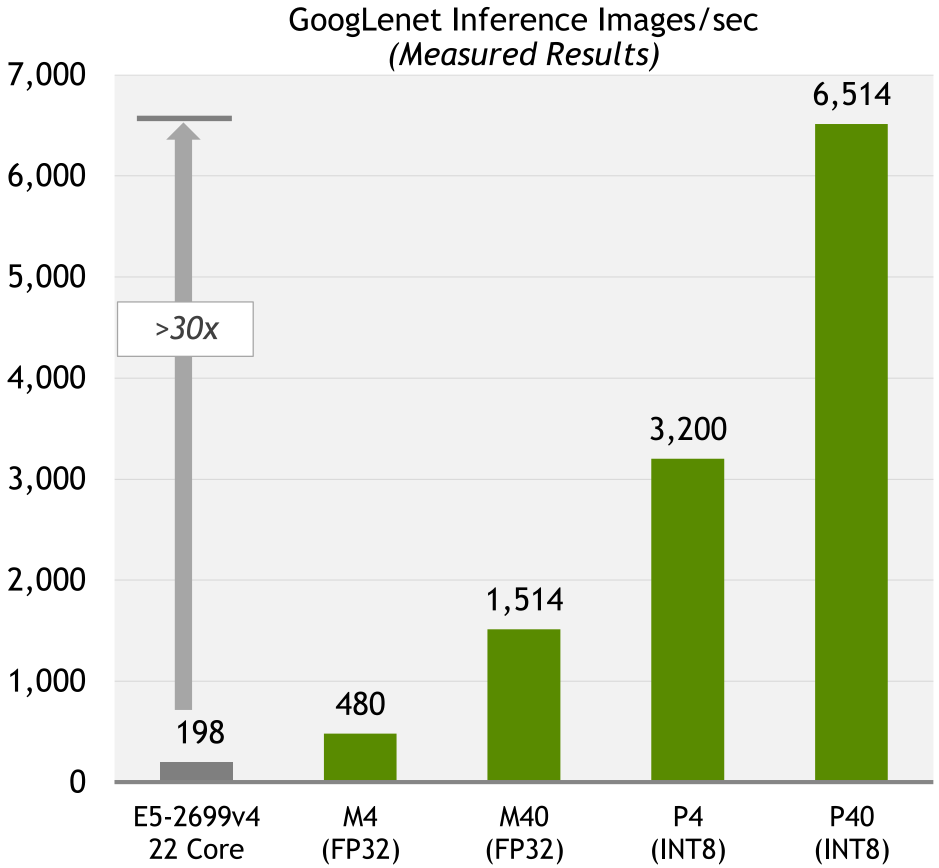


P40/P4 + INT8 + TENSORRT

Maximum deep learning inference efficiency



SOFTWARE	FEATURES
TensorRT v2	FP32, FP16, INT8
cuDNN v6	FP32, FP16, INT8



Measured results based on GoogLenet with batch size 128
Xeon uses MKL 2017 library with FP32, GPU uses TensorRT development ver.

NCCL: A multi-GPU collective communication library

DESIGN

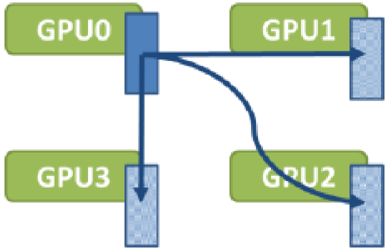
What is NCCL ?

- Optimized collective communication library between CUDA devices.
- Easy to integrate into any DL framework, as well as traditional HPC apps using MPI.
- Runs on the GPU using asynchronous CUDA kernels, for faster access to GPU memory, parallel reductions, NVLink usage.
- Operates on CUDA pointers. Operations are tied to a CUDA stream.
- Uses as little threads as possible to permit other computation to progress simultaneously.

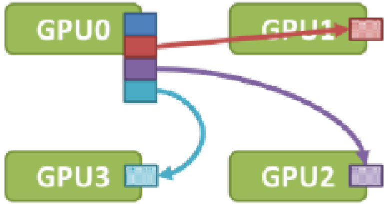
COLLECTIVE COMMUNICATION

Multiple senders and/or receivers

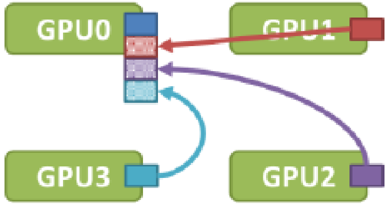
Broadcast



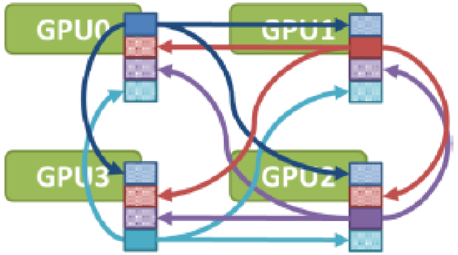
Scatter



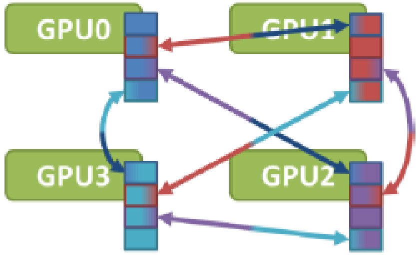
Gather



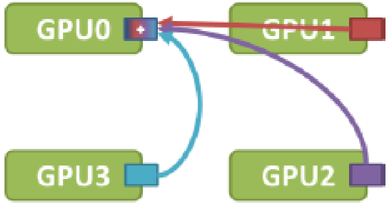
All-Gather



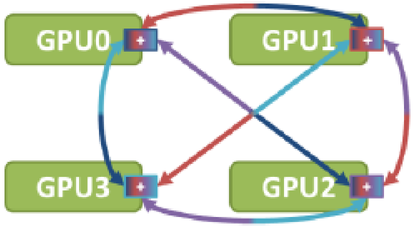
All-to-All



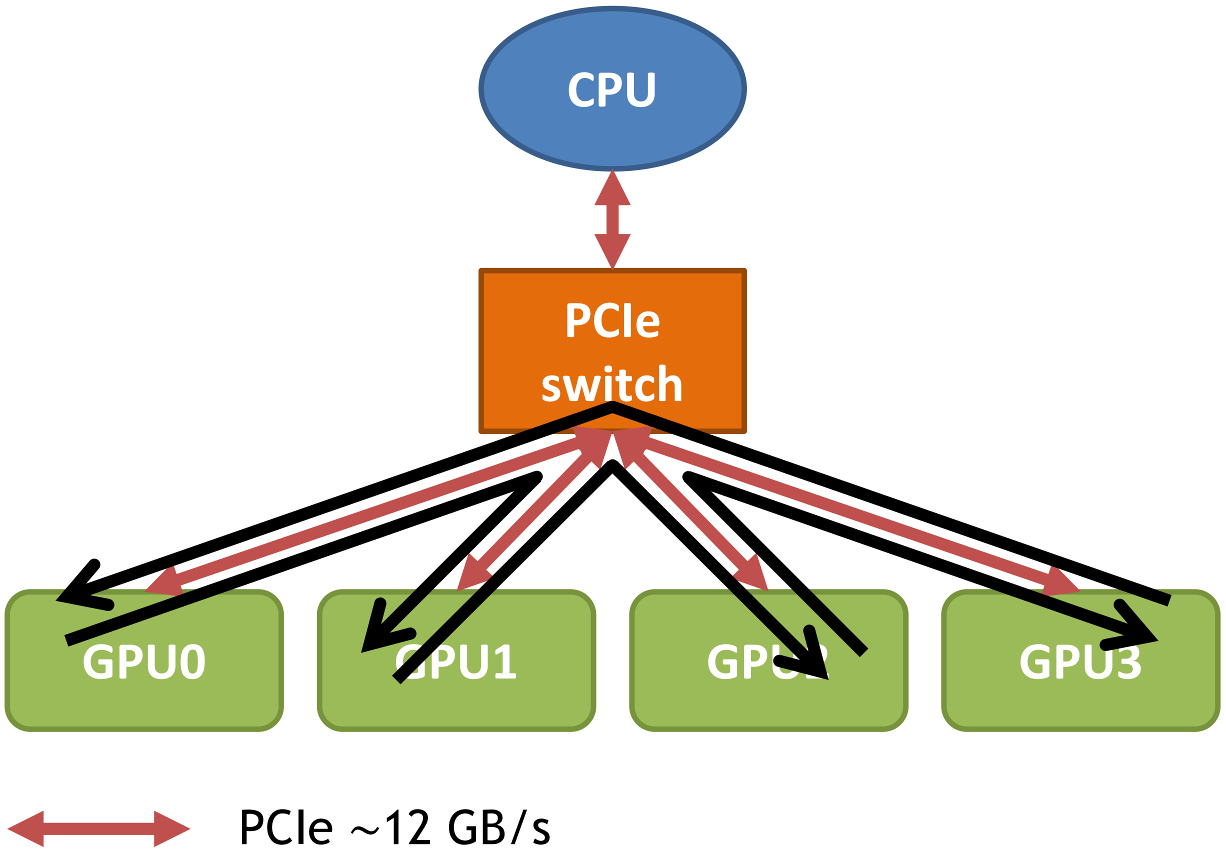
Reduce



All-Reduce



4-GPU PCIE TREE



+ exists today

+ all GPUs have P2P access

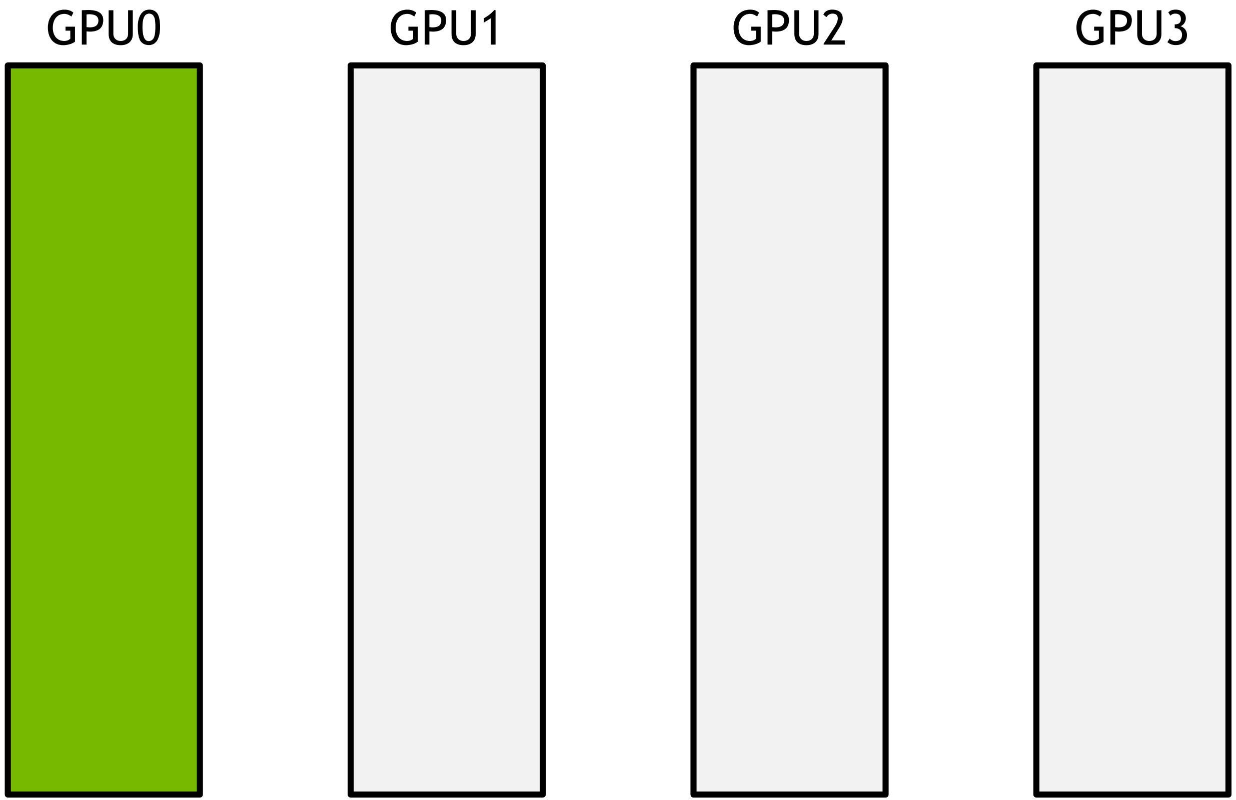
- PCIe is bottleneck

- no NVLink

can be thought of as unidirectional ring

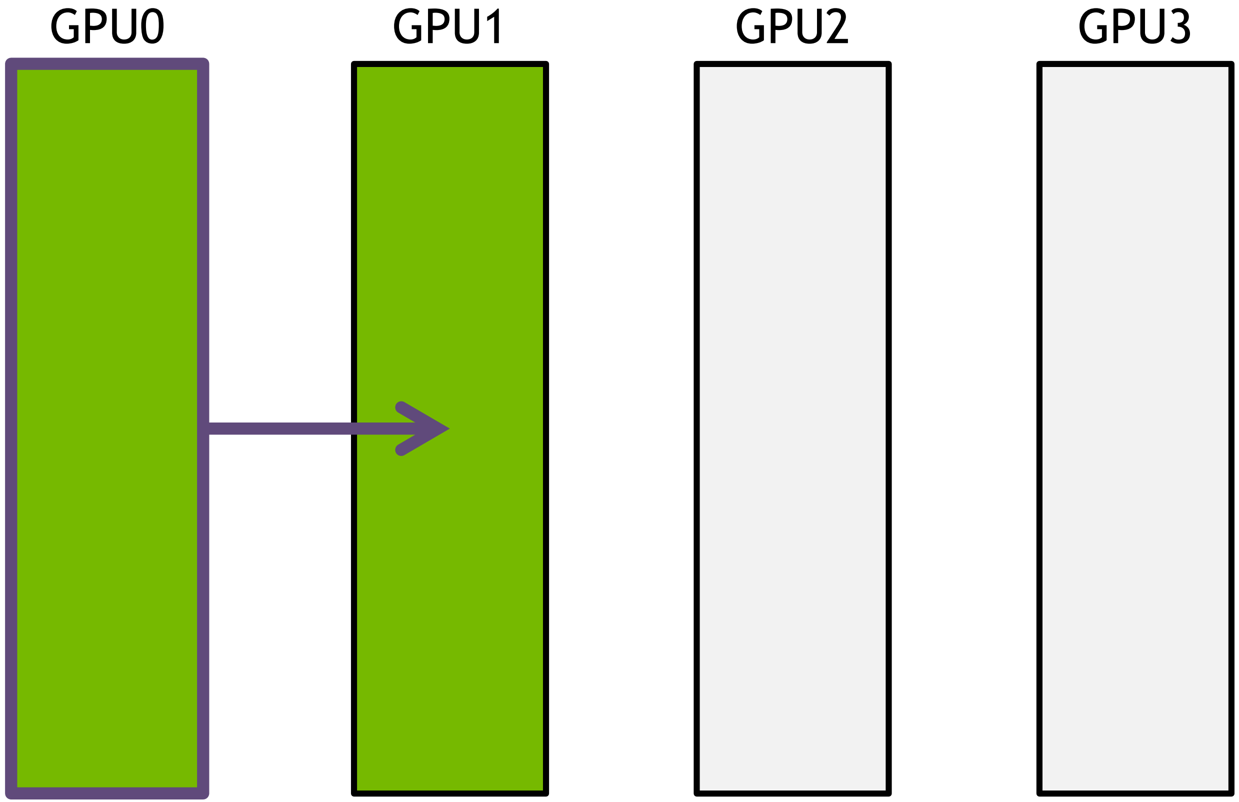
BROADCAST

with unidirectional ring



BROADCAST

with unidirectional ring



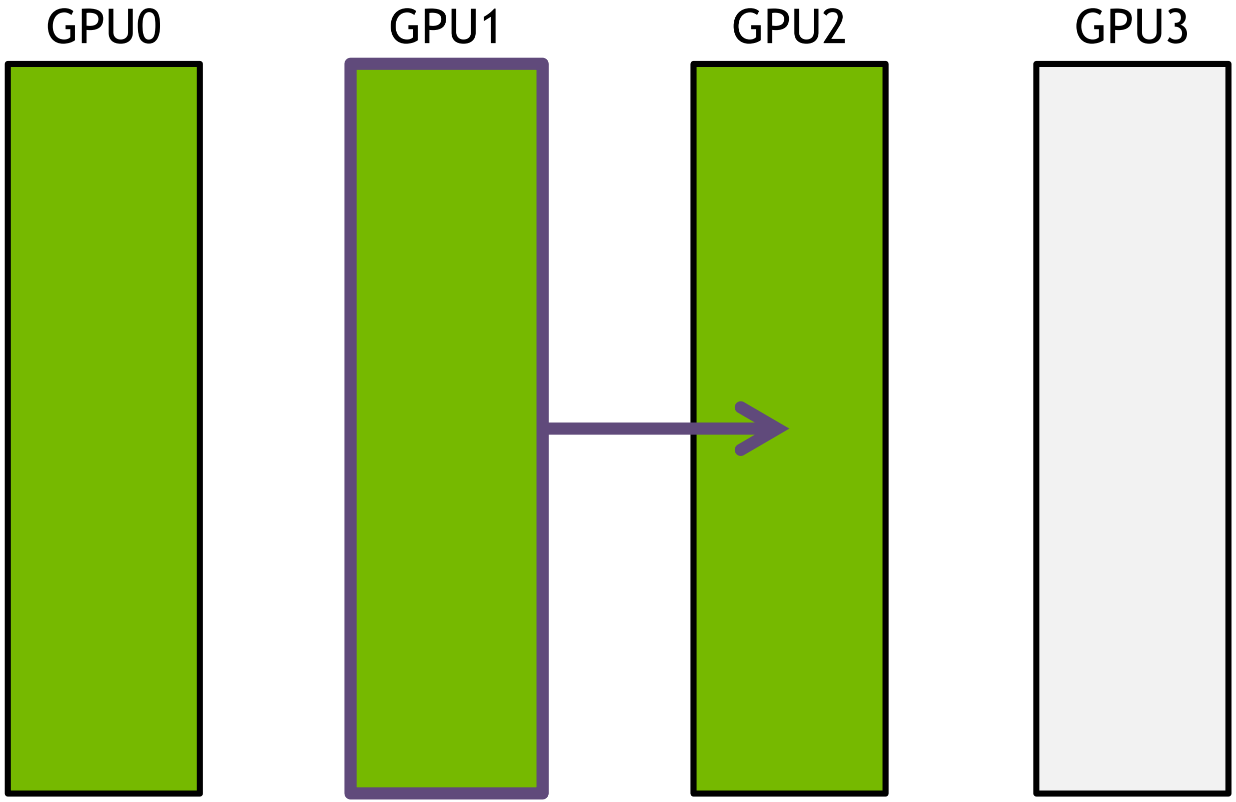
Step 1: $\Delta t = N/B$

N: bytes to broadcast

B: bandwidth of each link

BROADCAST

with unidirectional ring



Step 1: $\Delta t = N/B$

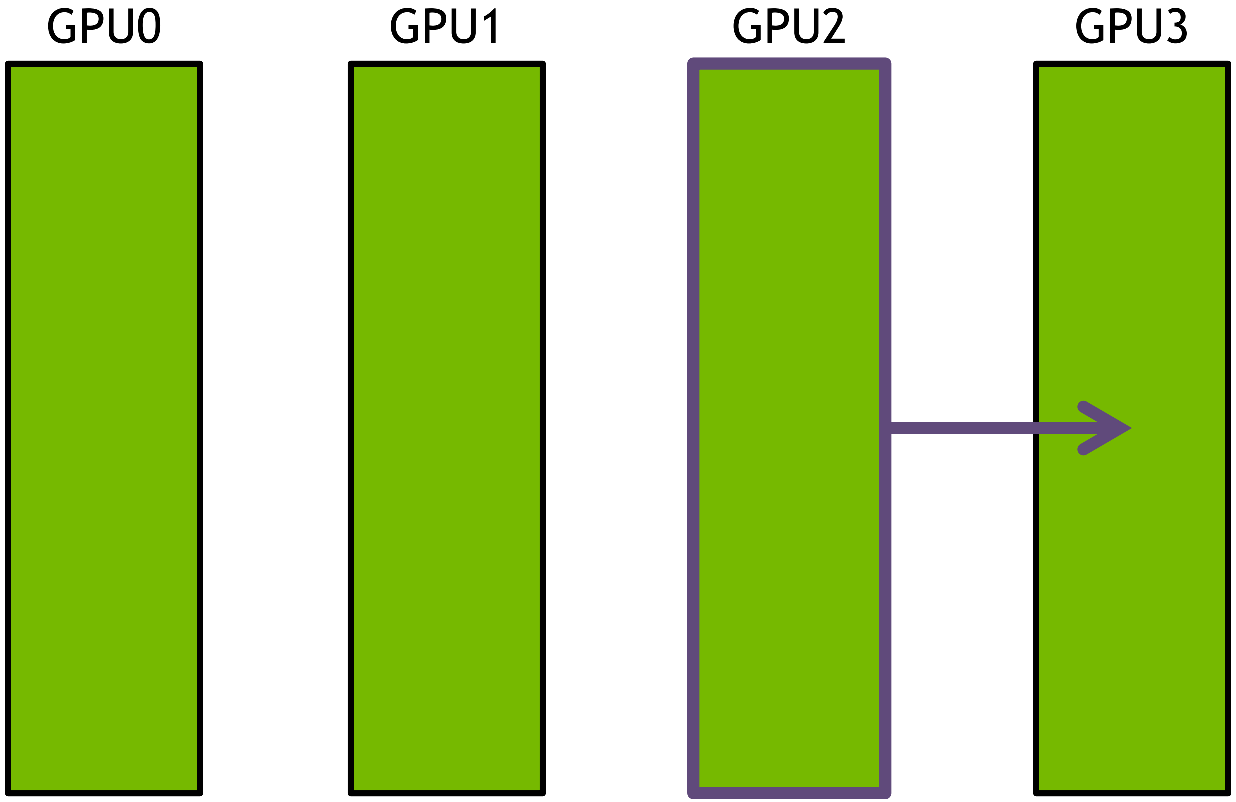
Step 2: $\Delta t = N/B$

N: bytes to broadcast

B: bandwidth of each link

BROADCAST

with unidirectional ring



Step 1: $\Delta t = N/B$

Step 2: $\Delta t = N/B$

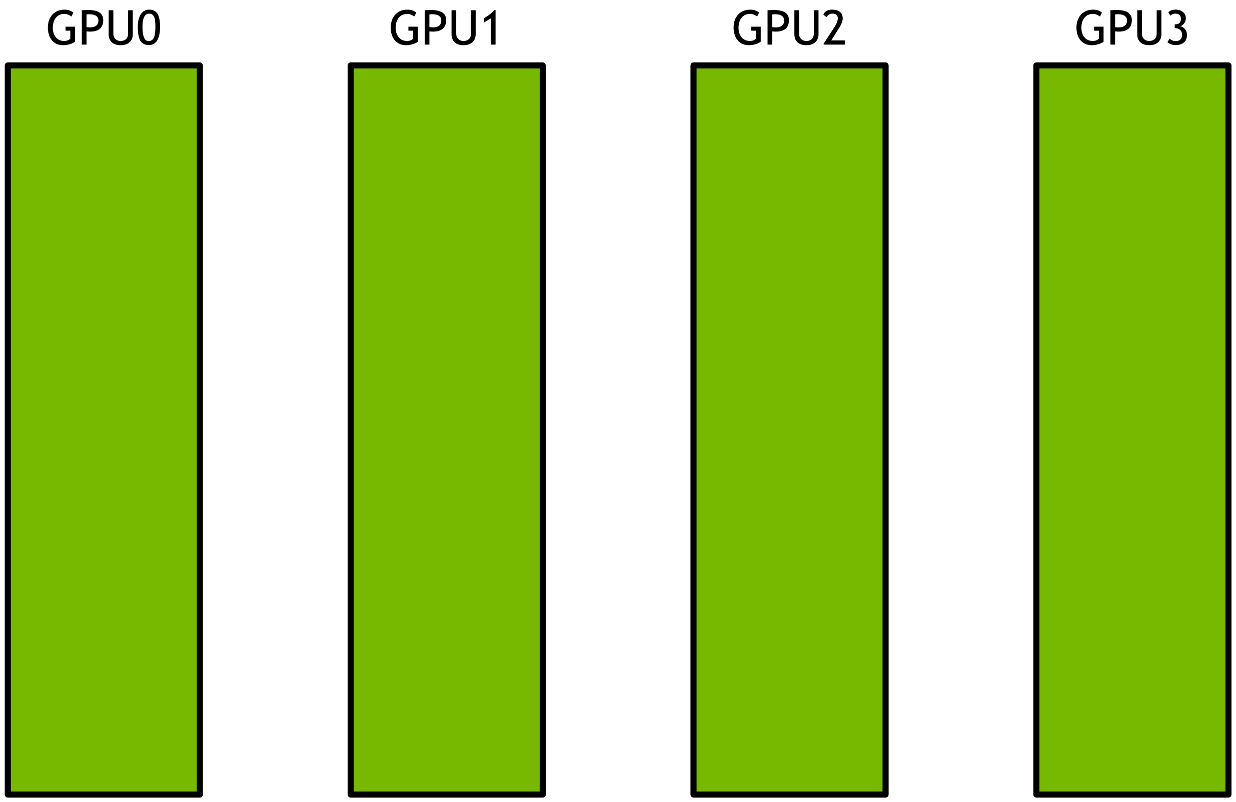
Step 3: $\Delta t = N/B$

N: bytes to broadcast

B: bandwidth of each link

BROADCAST

with unidirectional ring



Step 1: $\Delta t = N/B$

Step 2: $\Delta t = N/B$

Step 3: $\Delta t = N/B$

Total time: $(k-1)N/B$

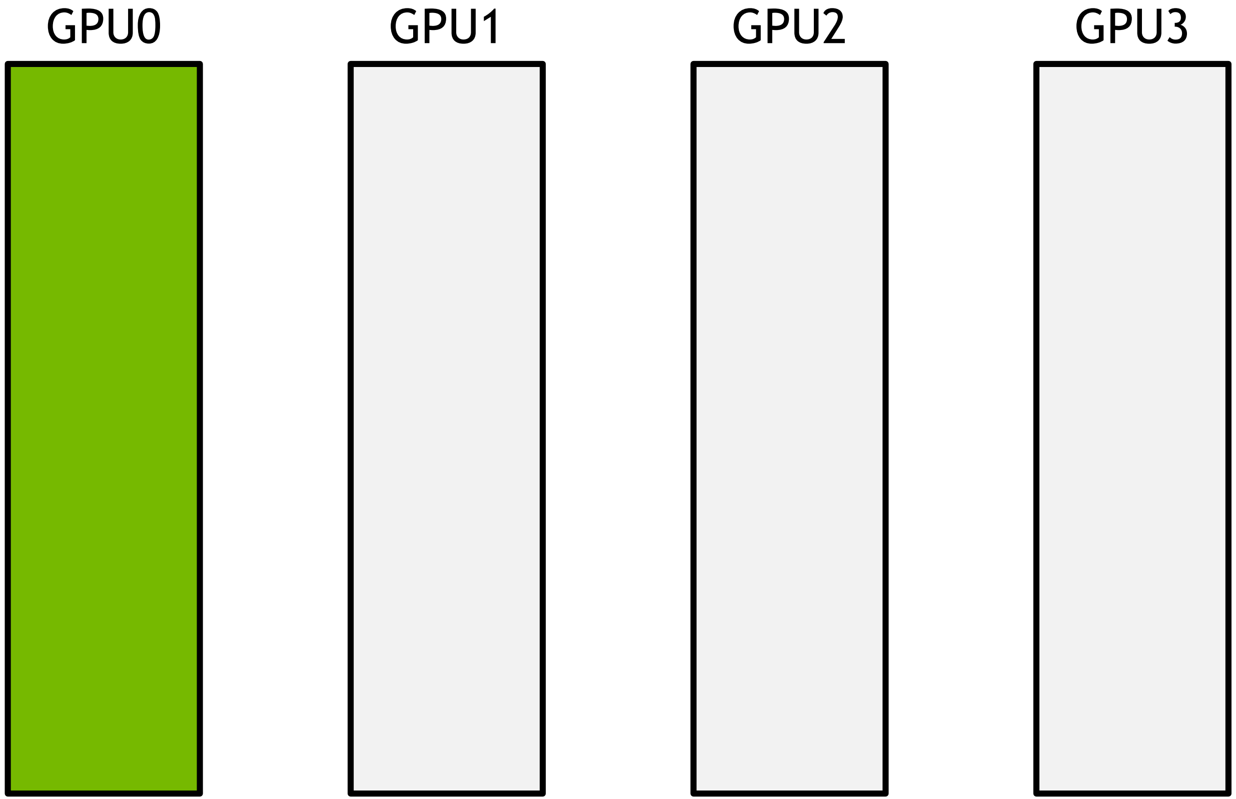
N: bytes to broadcast

B: bandwidth of each link

k: number of GPUs

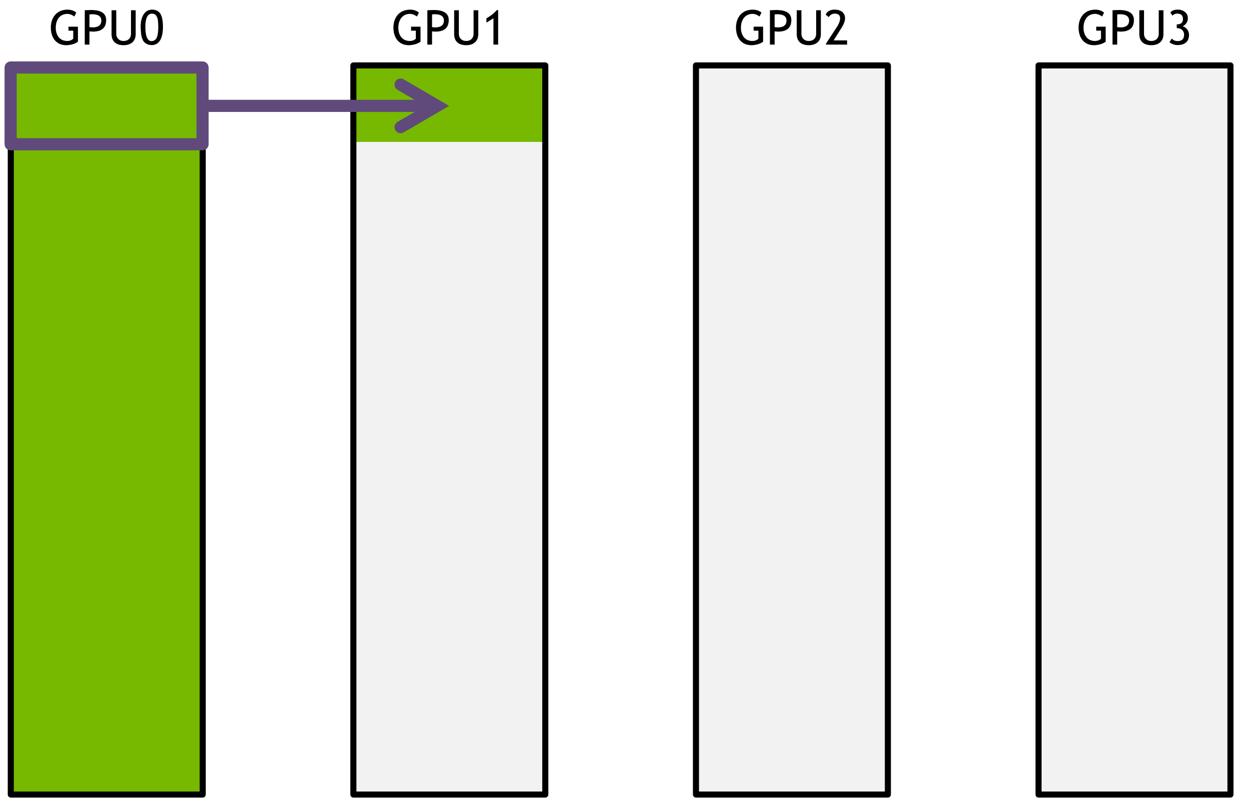
BROADCAST

with unidirectional ring



BROADCAST

with unidirectional ring

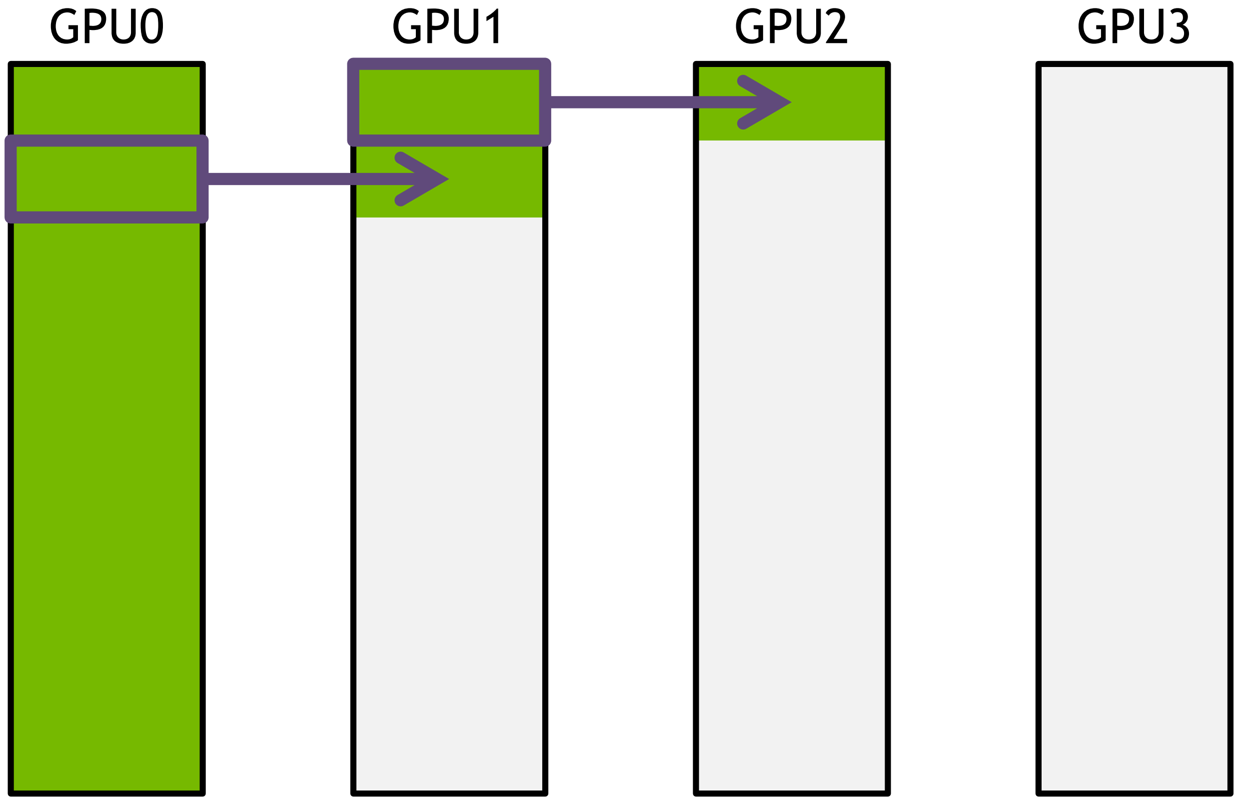


Split data into S messages

Step 1: $\Delta t = N/(SB)$

BROADCAST

with unidirectional ring



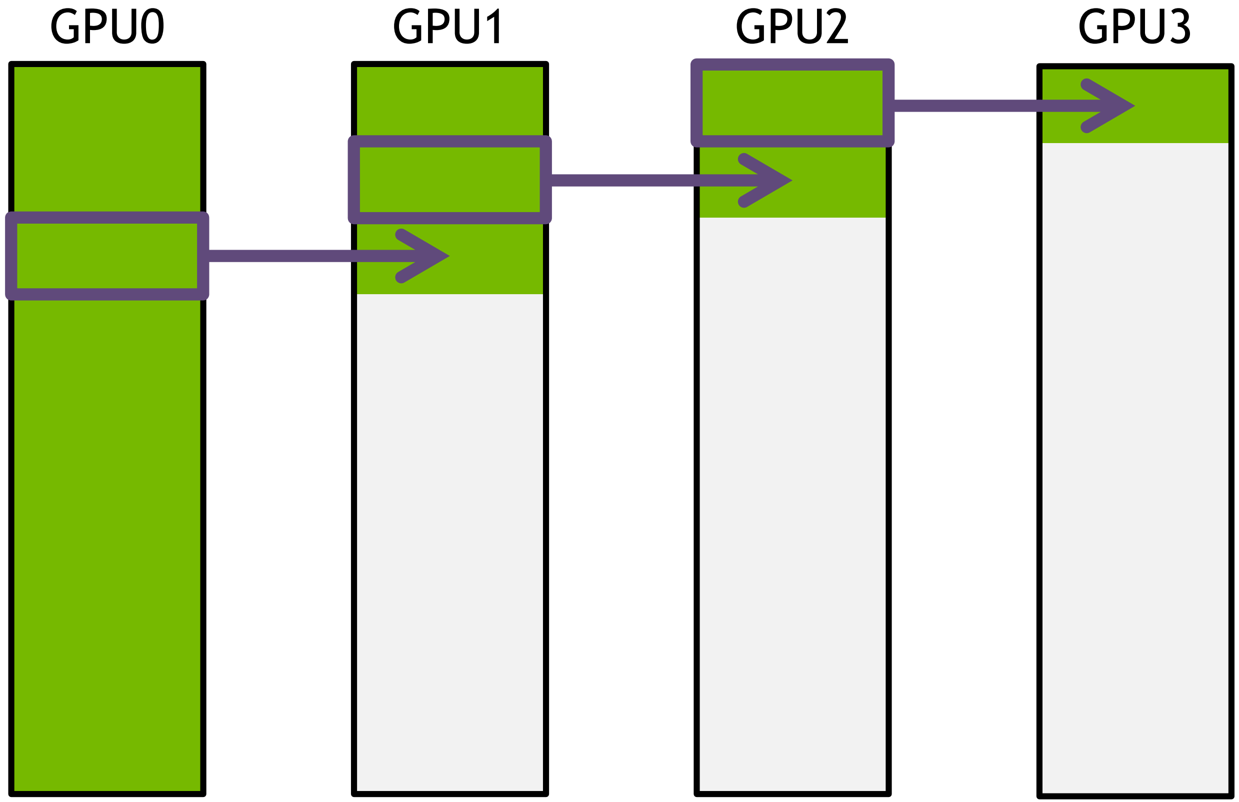
Split data into S messages

Step 1: $\Delta t = N/(SB)$

Step 2: $\Delta t = N/(SB)$

BROADCAST

with unidirectional ring



Split data into S messages

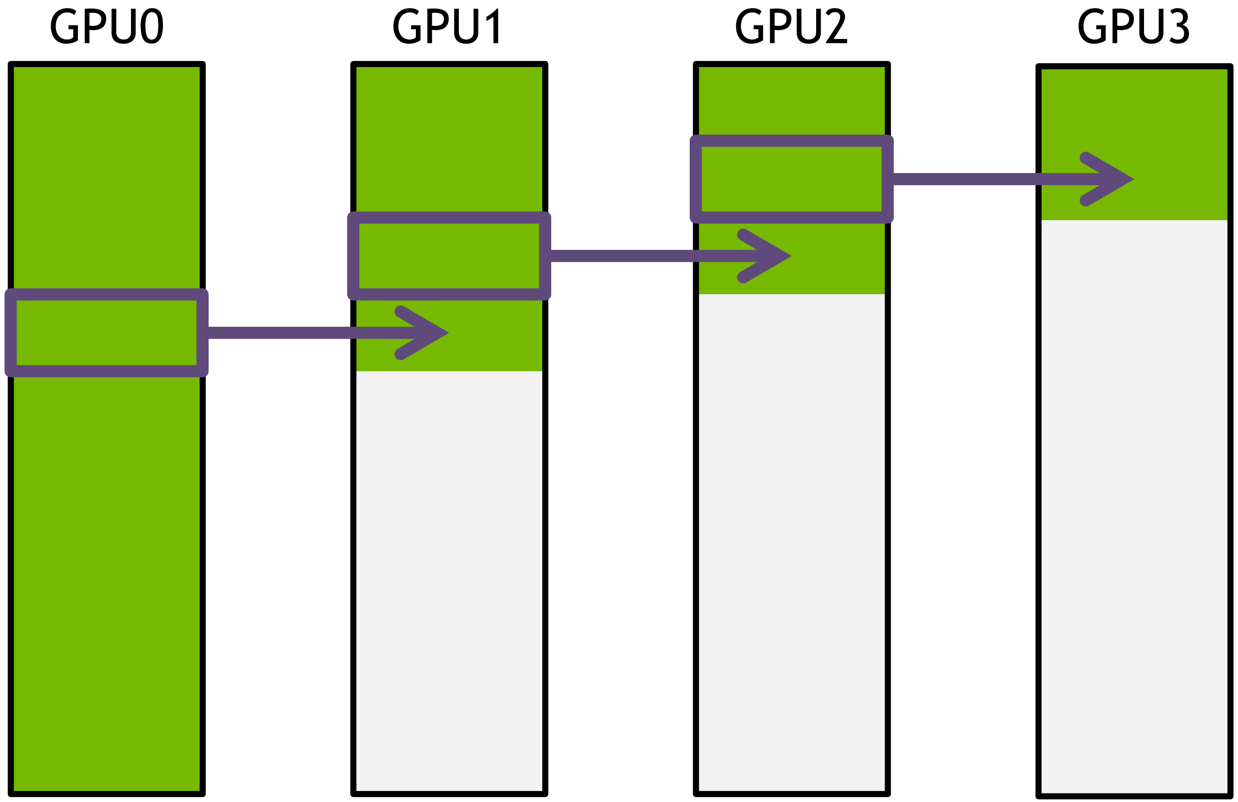
Step 1: $\Delta t = N/(SB)$

Step 2: $\Delta t = N/(SB)$

Step 3: $\Delta t = N/(SB)$

BROADCAST

with unidirectional ring



Split data into S messages

Step 1: $\Delta t = N/(SB)$

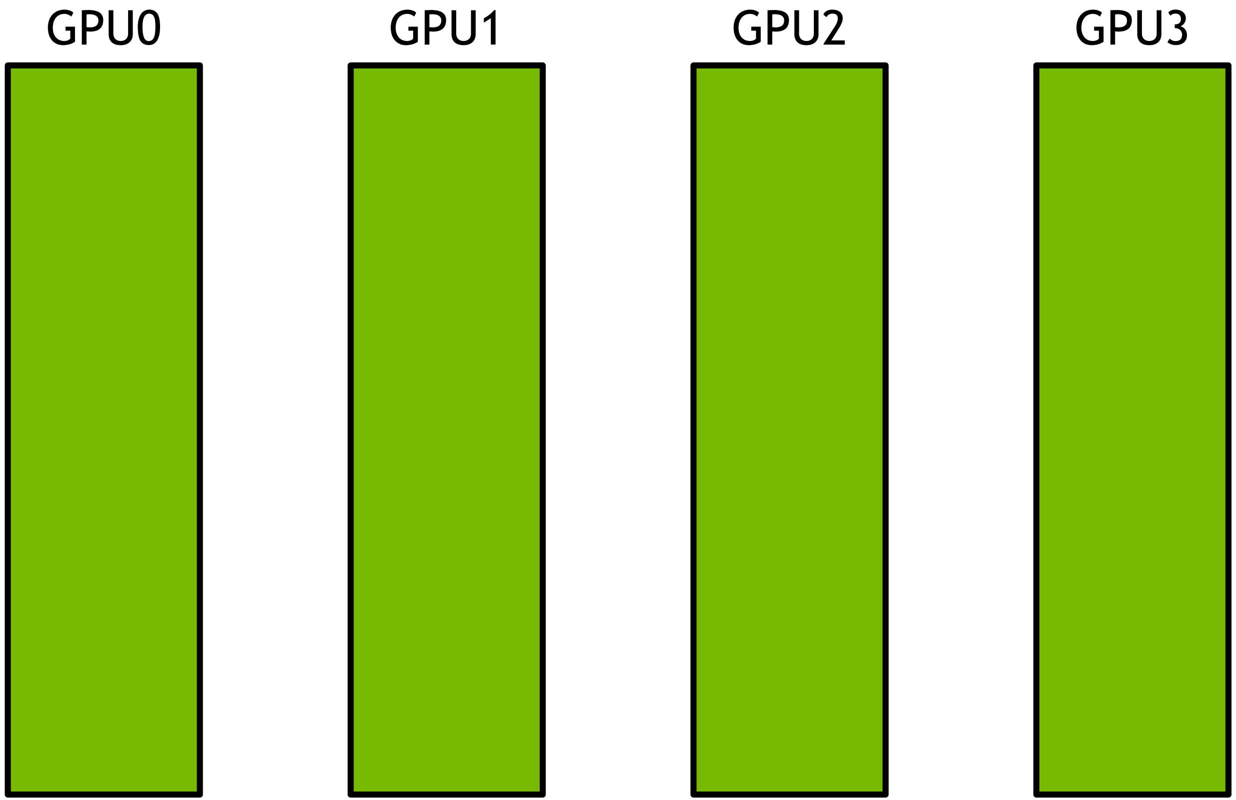
Step 2: $\Delta t = N/(SB)$

Step 3: $\Delta t = N/(SB)$

Step 4: $\Delta t = N/(SB)$

BROADCAST

with unidirectional ring



Split data into S messages

Step 1: $\Delta t = N/(SB)$

Step 2: $\Delta t = N/(SB)$

Step 3: $\Delta t = N/(SB)$

Step 4: $\Delta t = N/(SB)$

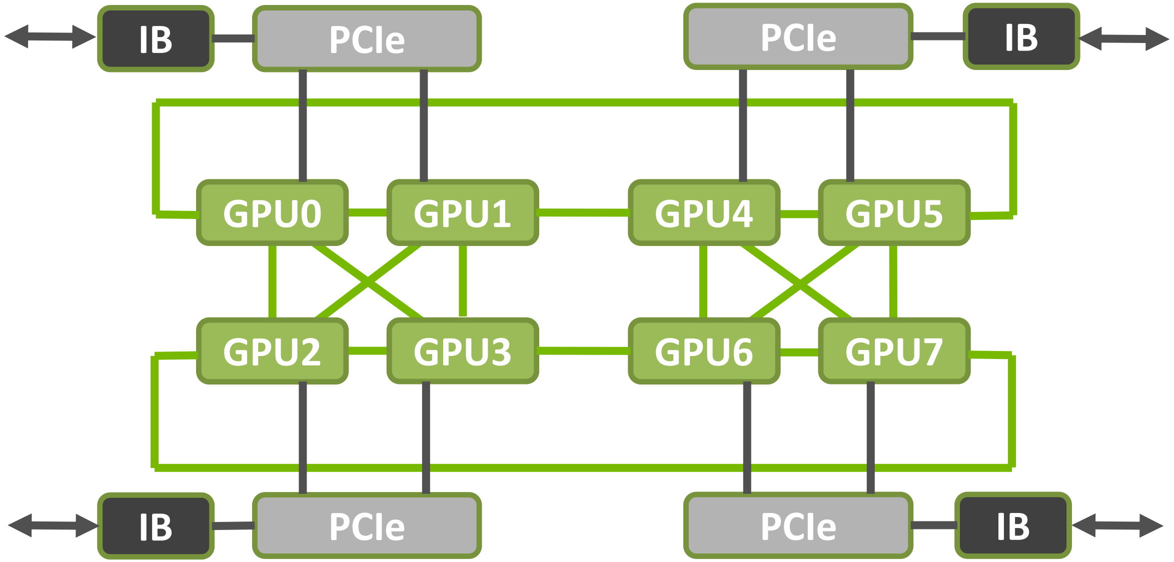
...

Total time:
$SN/(SB) + (k-2)\,N/(SB)$
$= N(S+k-2)/(SB) \rightarrow N/B$

NCCL 2.0

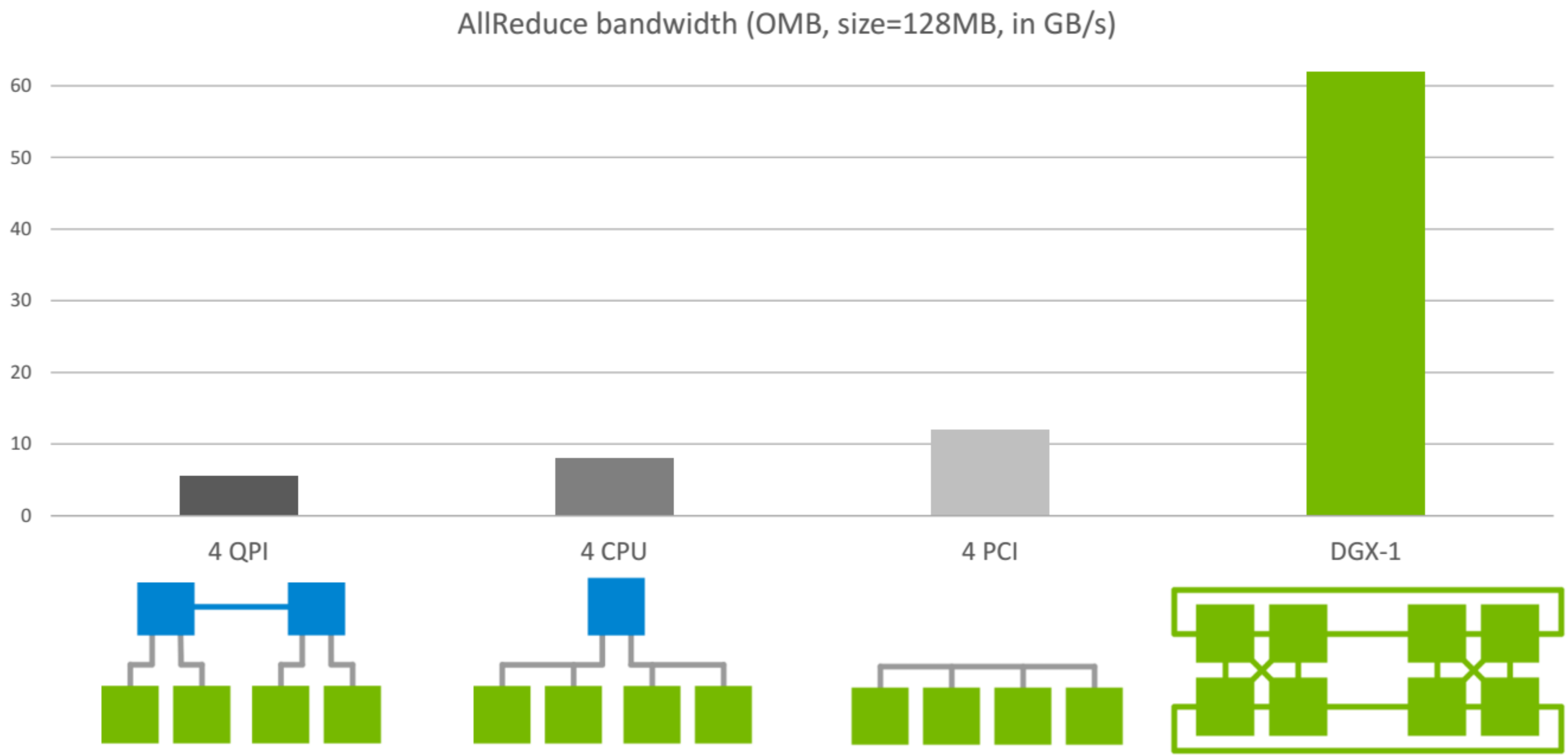
Inter-node communication

- Inter-node communication using Sockets or Infiniband verbs, with multi-rail support, topology detection and automatic use of GPU Direct RDMA.
- Optimal combination of NVLink, PCIe and network interfaces to maximize bandwidth and create rings across nodes.



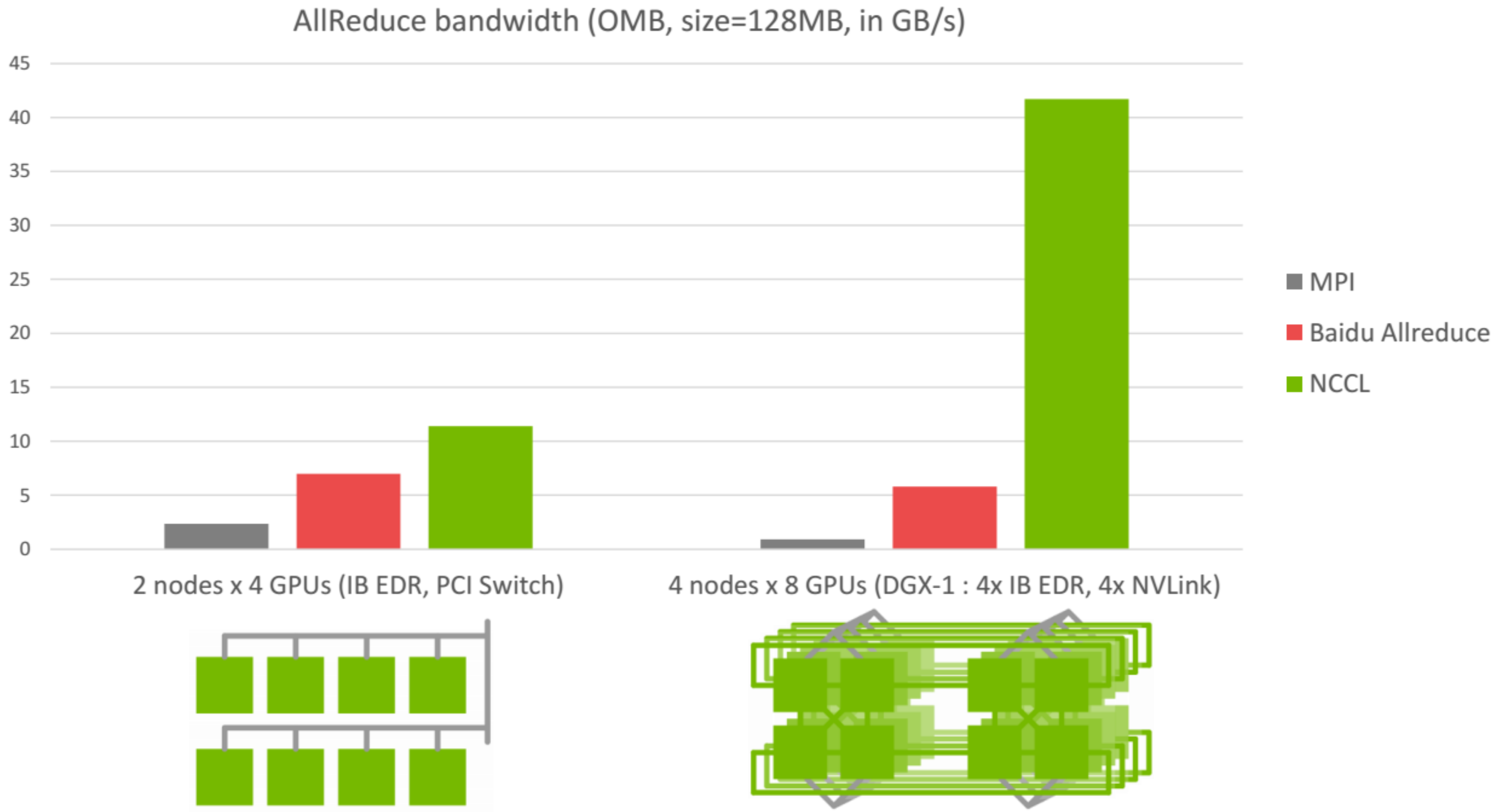
PERFORMANCE

Intra-node performance



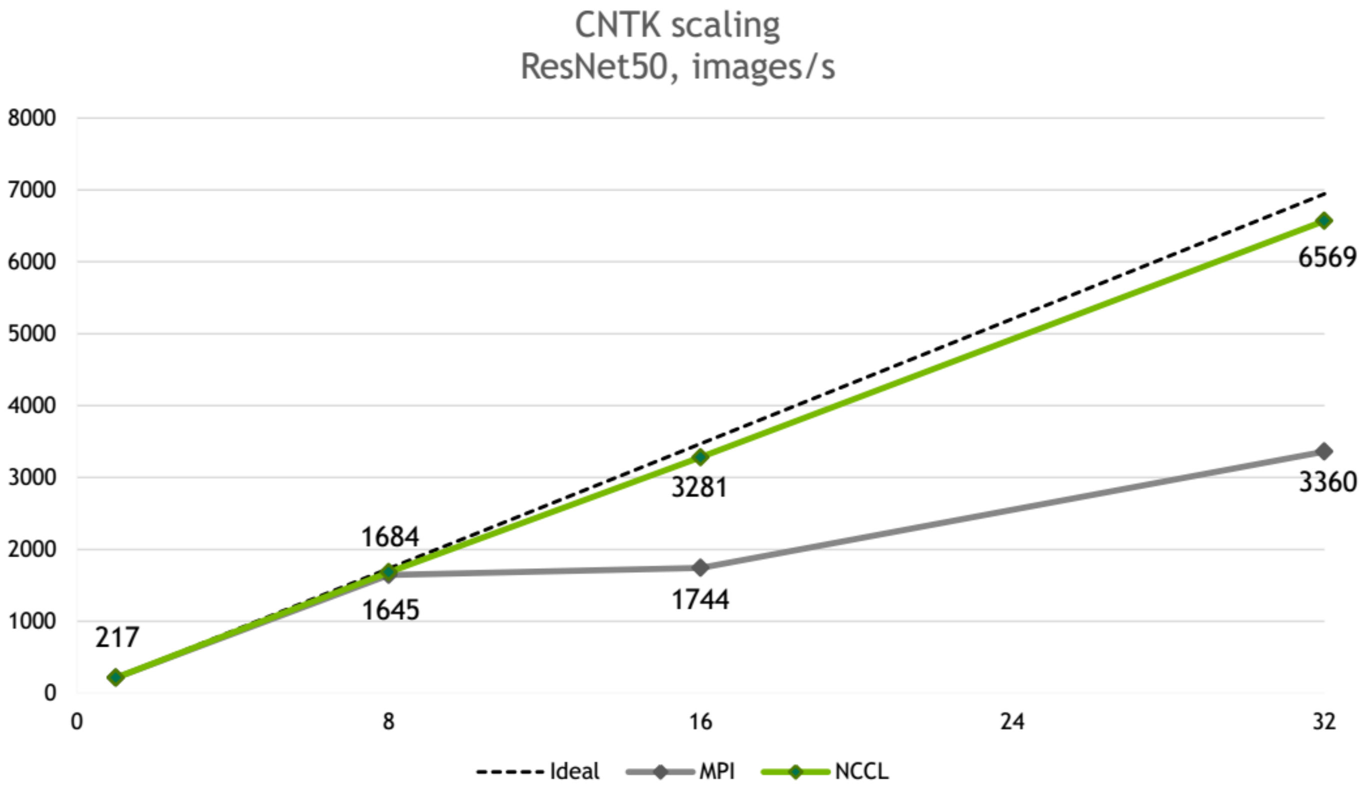
PERFORMANCE

Inter-node performance



PERFORMANCE

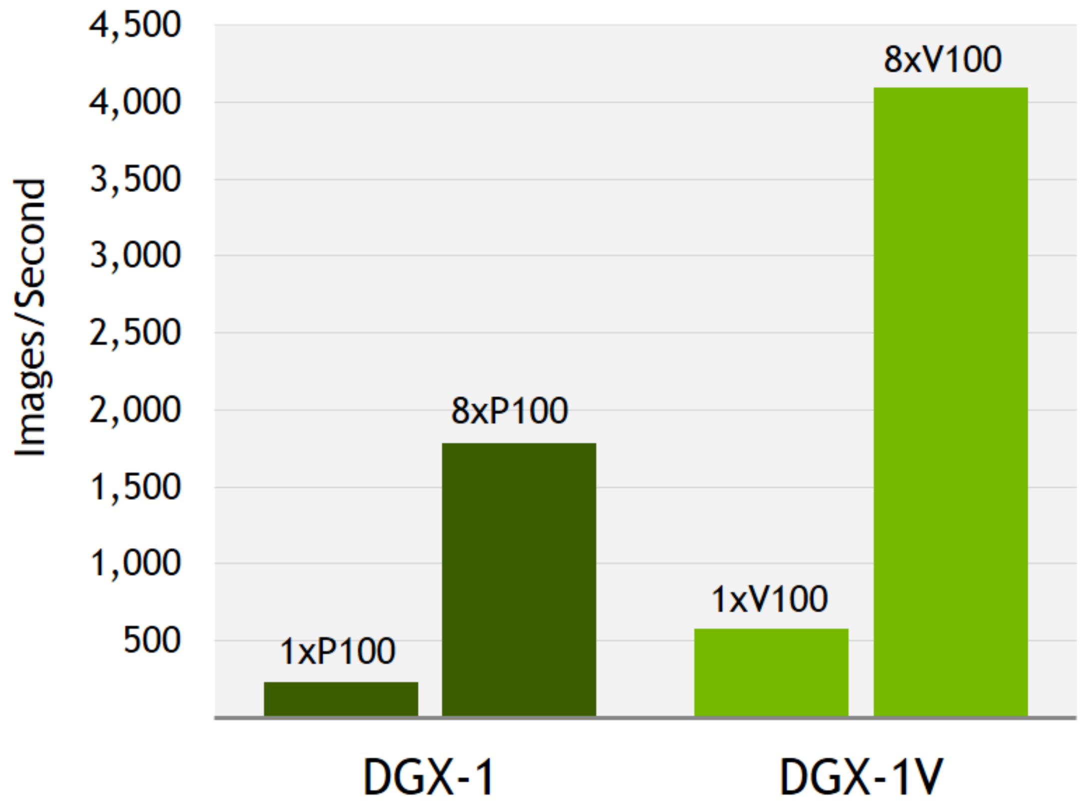
Near-Linear Multi-Node Scaling



Microsoft Cognitive Toolkit multi-node scaling performance (images/sec), NVIDIA DGX-1 + cuDNN 6 (FP32), ResNet50, Batch Size: 64

PERFORMANCE

7x Faster Training on DGX-1 vs. single GPU



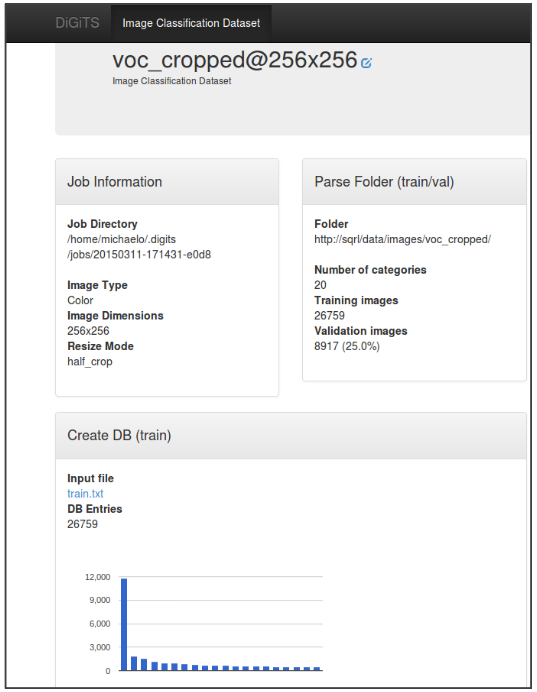
Caffe2 multi-GPU performance (images/sec) DGX-1 + cuDNN6 (FP32), DGX-1V + cuDNN 7 (FP16), Resnet50, Batch Size: 64

DIGITS

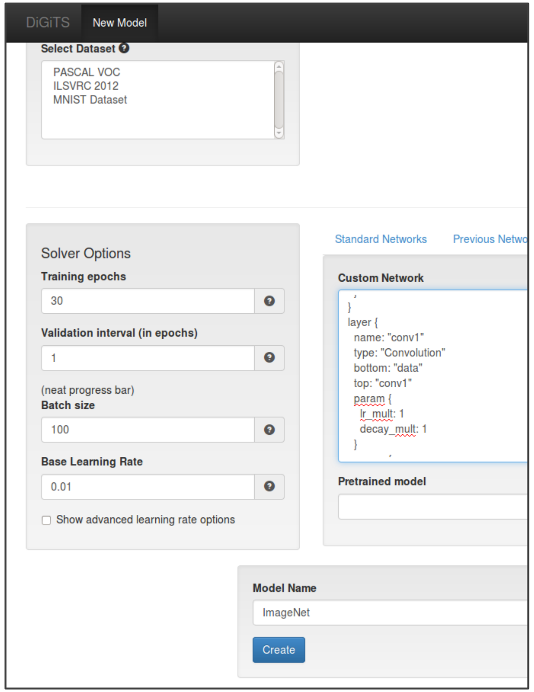
NVIDIA DIGITS

Interactive Deep Learning GPU Training System

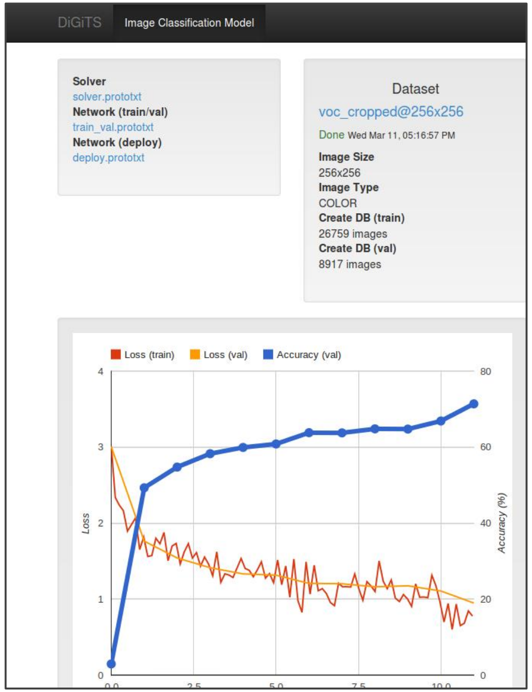
Process Data



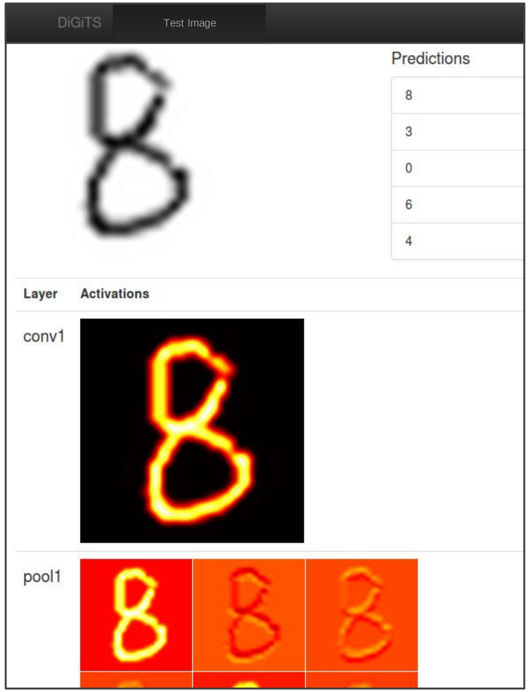
Configure DNN



Monitor Progress



Visualize Layers

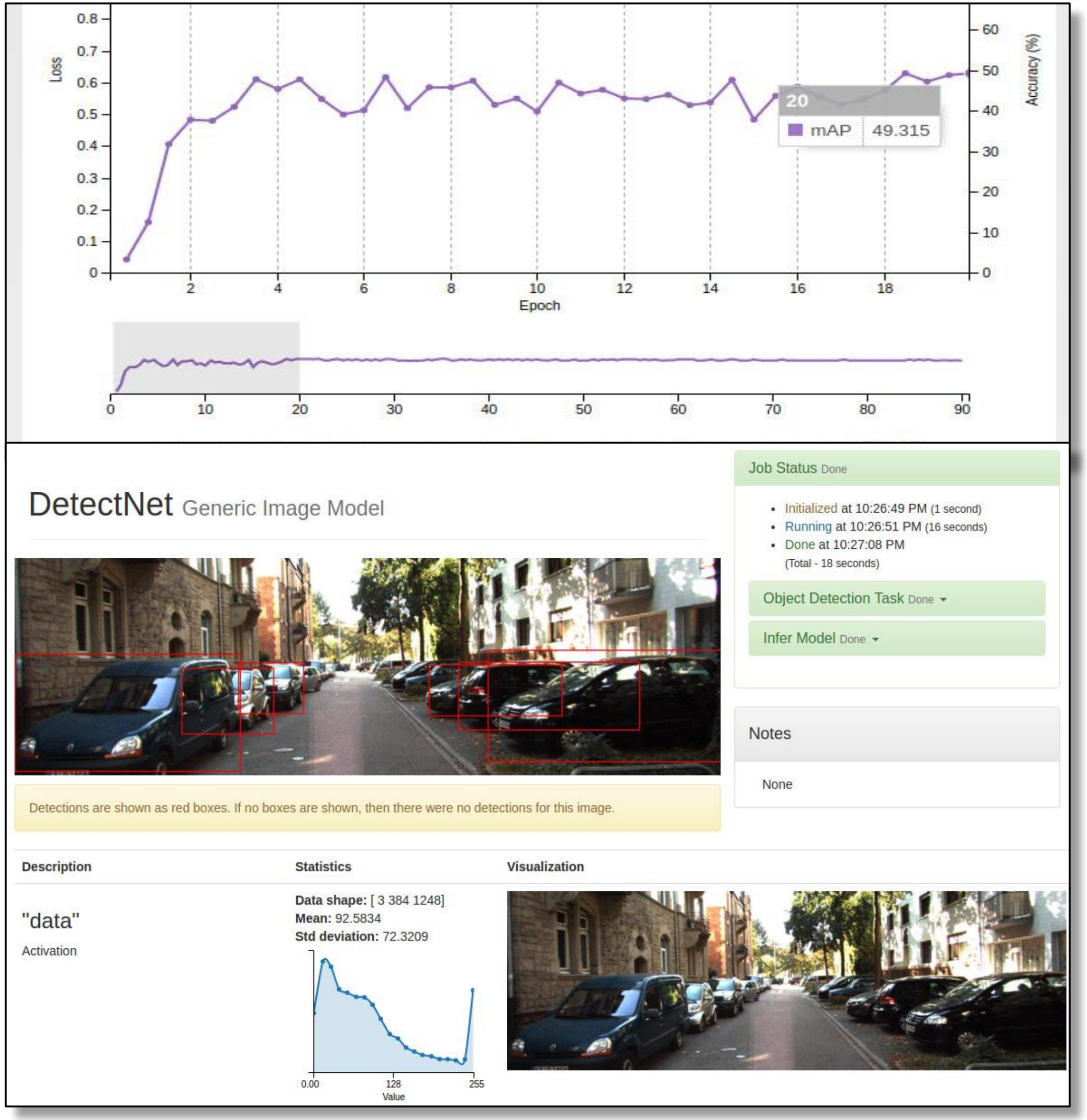


developer.nvidia.com/digits

DIGITS 4

Object Detection Workflow

- Object Detection Workflows for Automotive and Defense
- Targeted at Autonomous Vehicles, Remote Sensing



developer.nvidia.com/digits
github.com/NVIDIA/DIGITS

Reference

REFERENCES

- CUDA 9.0 RC can be downloaded from

  https://developer.nvidia.com/cuda-toolkit

- CUBLAS is delivered with CUDA 9.0 RC.
- cuDNN v7 can be downloaded from

  https://developer.nvidia.com/cudnn

- NCCL 2.0 can be downloaded from

  https://developer.nvidia.com/nccl

NVIDIA